VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Implementace komponenty konzolového serveru pro Virtlab

Implementation of Remote Management Console Access Server

2013 Matěj Děcký

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Matěj Děcký**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Implementace komponenty konzolového serveru pro Virtlab
Implementation of Remote Management Console Access Server

Zásady pro vypracování:

Cílem práce je reimplementace komponenty konzolového serveru a s ním související klientské aplikace pro řízený vzdálený přístup k management rozhraním prvků Virtlabu. Oproti stávajícímu řešení bude nové jednodušší o systém zakázaných příkazů na straně serveru. Naopak rozšíření zaznamená klientská strana, kde vznikne plnohodnotná multiplatformní aplikace s upraveným komunikačním protokolem pro jednoduchou a bezpečnou autentizaci uživatele a řízení přístupu k prvkům rezervací. Stávající řešení webového appletu bude i nadále serverem podporováno.
1. Seznamte se stávajícím řešením konzolového serveru a analyzujte požadavky z něj plynoucí na kompatibilitu s webovým appletem.
2. Navrhněte a implemetujte nové řešení konzolového serveru a klientské aplikace.
3. Výsledné řešení řádně otestujte.
4. Zhodnoťte dosažené výsledky, koncept a funkčnost celého řešení.

Seznam doporučené odborné literatury:

BECK, Michael. Linux kernel internals. 2nd. USA: Addison-Wesley, 1998, 480 s. ISBN 9780201331431.
BENVENUTI, Christian. Understanding Linux network internals. USA: O Reilly Media, Inc., 2006, 1035 s. ISBN 9780596002558.
SMITH, Roderick W. Advanced Linux networking. USA: Addison-Wesley, 2002, 752 s. ISBN 9780201774238.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Martin Milata**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 25. dubna 2013 .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, hlavně pak vedoucímu práce, který byl zdrojem užitečných rad při psaní a také mé múze, která mi pomohla s vytyčením bodů pro nerušené psaní této diplomové práce.

Abstrakt

Cílem práce je reimplementace komponenty konzolového serveru a s ním související klientské aplikace pro řízený vzdálený přístup k management rozhraním prvků Virtlabu. Oproti stávajícímu řešení bude nové jednodušší o systém zakázaných příkazů na straně serveru. Naopak rozšíření zaznamená klientská strana, kde vznikne plnohodnotná multiplatformní aplikace s upraveným komunikačním protokolem pro jednoduchou a bezpečnou autentizaci uživatele a řízení přístupu k prvkům rezervací. Stávající řešení webového appletu bude i nadále serverem podporováno.

Klíčová slova: C++, virtlab, qt, databáze diplomová práce

Abstract

Main goal of this thesis is reimplementation of console server a with that creation of client application for remote access to management of Vitlab devices interfaces. New solution will be easier for system of forbidden commands in server. Client site will be richer of new multiplatform application with modified communication protocol for easier and safe authentication of user and controling access to devices.

Keywords: c++, virtlab, qt, databse, master thesis

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
ASSSK	–	Automatizovaný systém správy síťových konfigurací
BASH	–	Bourne again shell
GUI	–	Graphical user interface
IP	–	Internet protocol
LDAP	–	Lightweight Directory Access Protocol
OSI	–	Open Systems Interconnection
PHP	–	Professional home pages
SHA	–	Secure hash algorythm
SSL	–	Socket secure layer
SQL	–	Structured query language
TCP	–	Transmission control protocol
VLAN	–	Virtual local area network

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

Virtuální laboratoř počítačových sítí (zkráceně Virtlab) je projekt sloužící k tomu, aby bylo možno přistupovat k prvkům předem nastavené síťové topologie, kterými laboratoř disponuje, skrze Internet. Celou topologii je možné si rezervovat přes webový portál, kde je možno si zvolit jak čas, tak i spolupracovníky na projektu. Virtlab sloužil k výuce zejména počítačových sítí. K přístupu ke vlastním síťovým prvkům topologie, ať už jde o směrovače, přepínače či pracovní stanice, slouží konzolový server. Tento konzolový server je službou, která slouží jako mezivrstva mezi přístupem k samotným konfiguračním rozhraním daných zařízení, ať už jde o konzoli zařízení nebo unixový shell. Reimplementace tohoto serveru je cílem této diplomové práce.

2 Popis virtuální laboratoře

2.1 Vznik projektu

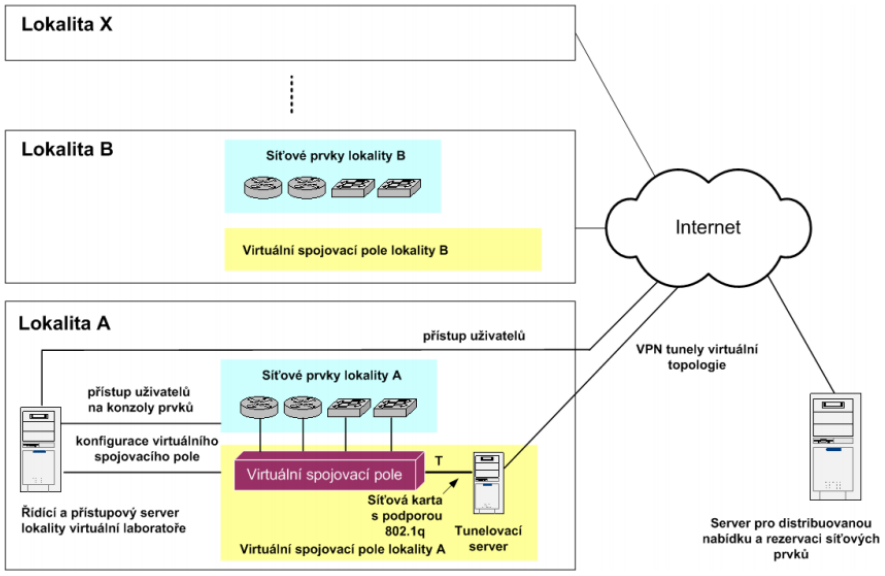
Projekt Virtlab byl prvně iniciován v roce 2005 v diplomové prácí Pavla Němce, jehož náplní bylo řešení problému zpřístupnění síťových prvků pro studenty CNAP (Cisco Networking Academy), do jehož programu se naše fakulta začlenila v roce 1999. Tato varianta však ještě neumožnovala dynamické propojování topologií. Toho bylo docíleno až návrhem ASSSK(Automatizovaný Systém pro Správu Síťových Konfigurací), neoficiálně pojmenovaný TataBazmek nebo Bazmek, ing. Davidem Saidelem v jeho diplomové práci. Od jeho prvního použití již byl mnohokrát upraven a existuje současně ve verzi 2+, která byla naposledy upravena Petrem Havlíčkem. Když v roce 2005 Virtlab vznikl, existovala pouze jeho nedistriubovatelná varianta. To se změnilo v roce 2007, kdy byla v diplomové práci Jana Vavřícka a Tomáše Hrabálka popsána její distribuovatelná varianta. Jedná se o myšlenku více virtuálních lokalit, logicky spojených v síti Internet a navzájem si poskytující zařízení, jejiž pořízení by mohlo být pro jednotlivé lokality nákladné. Samotný Virtlab byl mnohokrát vylepšen a upraven v mnoha diplomových či bakalářských pracích, např. Automatizace hodnocení konfigurací od Zdeňka Filipce nebo Distribuované spojovací pole od Václava Bortlíka, jejiž celý seznam může být nalezen na wiki stránce projektu Virtlab.

2.2 Architektura distribuované verze projektu Virtlab

Jak už bylo řečeno, distribuovaná verze projektu Virtlab se skládá z tzv. lokalit. Lokalita je struktura, skládající se z softwarových a jiných hardwarových prostředků. Každá takováto struktura může komunikovat s jinými strukturami za účelem sdílení vlastních fyzických zařízení. Lokalita je tvořena třemi prvky:

- **Řídící webová aplikace** - webová aplikace napsaná v php, přes kterou je vedena administrace systému a přes kterou je umožněno jednotlivým uživatelům vytvoření, editace či jiná práce s rezervacemi a správa vlastního účtu.
- **Serverová část** - mezivrstva mezi webovou aplikací a vlastními zařízeními. Jejím účelem je umožnění samotné komunikace mezi uživatelem a zařízením. Dále spravuje data uživatelů v databázi a spravuje a půjčuje zařízení pro rezervace. Konzolová část bude blíže popsána v kapitole 4.
- **Zařízení** - jedná se o fyzické síťové prvky nutné pro běh Virtlabu. Například zařízení ASSSK nebo MOXA karty.

Na obrázku 1 je zakresleno schéma takovéto distribuované varianty. Z obrázku je jasně vidět, že může obsahovat několik lokalit, kdy každá z lokalit má vlastní server, jakousi část propojující jednotlivá zařízení a všechny tyto lokality jsou spolu propojeny tunelovacím serverem.



Obrázek 1: Distribuovaná verze Virtlabu

2.3 Konzolová část Virtlabu

Konzolová část virtlabu je jednou ze součástí serverové části. V současné verzi je přístupná pro uživatele buď přes webový java applet nebo přes službu telnet, kdy se je následnou nutností kopírování poskytnutých textových řetězců. Samotnou fukncí konzolového serveru jsou následující věci:

- **Spojení uživatele se zařízením** - základní komunikace zařízení - uživatel. Nutno autentizace uživatele.
- **Spojení dvou konzolových serverů** - přenost dat mezi lokalitami přes již vytvořený tunel. Autentizaci provádí lokální konzolový server (ten, ke kterému se připojil uživatel).
- **Administrace spojení** - nutná možnost administrátora nahlížet do logů, zobrazit seznam aktivních spojení a případně i možnost jejich terminace.

3 Síťová komunikace

3.1 IP protokol

Internetový protokol je jednou ze sady protokolů pracujících na třetí (síťové) vrstvě modelu vrstvené síťové architektury ISO/OSI používaných v počítačových sítích a internetu. Slouží k zasílání datagramů skrze hranice lokálních sítí. Její návrh umožňuje propojování těchto počítačových sítí a v podstatě je základním kamenem Internetu. IP má za úkol doručování paketů ze zdrojového sítového rozhraní k cílovému výhradně na základě IP adresy. Díky tomu IP definuje datagramové struktury, které zapouzdřují cílová data. Definuje také metody adresace těchto datagramu se zdrojovou a cílovou adresou. První velkou verzí IP protokolu je Internet protokol verze 4 (IPv4), který je dodnes dominantním protokolem v Internetu. Jejím následníkem je Internet protokol verze 6 (IPv6).

3.1.1 Konstrukce IP datagramu

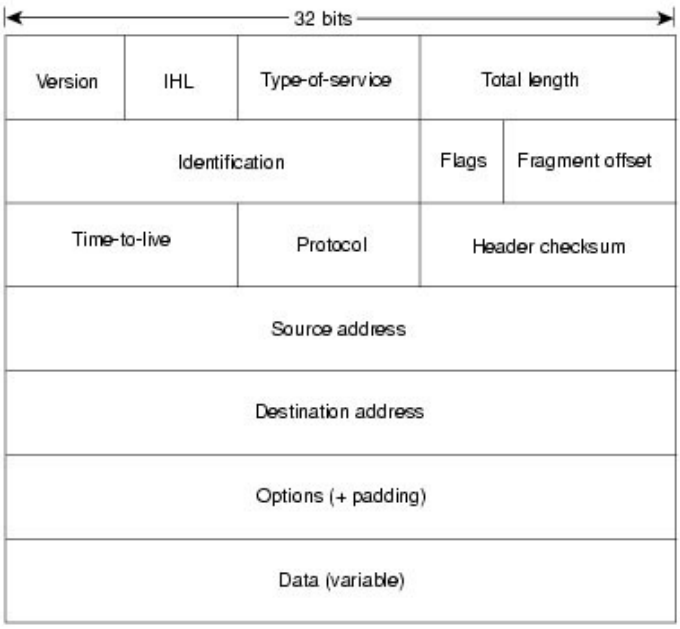
IP datagram je tvořen dvěmi částmi: hlavičkou a vlastním tělem (daty). Hlavička obsahuje zdrojovou a cílovou IP adresu, což je unikátní identifikátor každého rozhraní přímo připojeného k síti internet. Velikost adresy se liší podle verze daného IP protokolu. Pro IPv4 je to 32 bitů a pro IPv6 je to 128 bitů. Dále jsou v hlavičce obsažena metadata, označující verzi protokolu dané hlavičky a různé informace o datech a o chování datagramu v síti. Tělo je tvořeno vlastními daty, které se přeposílají. Tato metoda přidání hlavičky k datům se nazývá zapouzdření. Ukázka hlavičky je na obrázku 2.

3.1.2 Spolehlivost IP

Návrh IP předpokládá, že síťová infrastruktura je nespolehlivá a neustále se v ní mění propojení uzlů. V zájmů snížení složitosti sítě se využívá princip konec-konec. Proto je tedy veškerá logika prováděna jen na koncích přenosu nebo co nejblíže cíli. Jako důsledek tohoto návrhu, IP protokol poskytuje jen přenos a obecně je charakterizován jako nespolehlivý. Je tedy označen za bezespojový protokol narozdíl od protokolů spojově-orientovaných. Při přenosu může nastat ztráta datagramu, poškození přenášených dat, několikanásobné doručení či doručení v různém pořadí. IPv4 zajišťuje základní ochranu hlaviček, kdy každá hlavička v sobě obsahuje kontrolní součet dat v hlavičce a pokud některý z uzlů detekuje rozdíl mezi hlavičkou a jejím kontrolním součtem, automaticky data zahodí.

3.2 TCP

Transmition Control Protokol je jedním ze dvou základních protokolů sady IP a je tak zásadní, že celá sada se nazývá TCP/IP. TCP zajišťuje spolehlivý, uspořádaný a bezchybný tok dat mezi rozhraními, připojených k počítačové síti. Aplikace, které nevyžadují spolehlivost mohlou místo TCP použít protokol UDP (user datagram protocol), který je narozdíl od TCP bezespojový, ale jeho výhodou je rychlejší odezva. Oba protokoly jsou v modelu OSI na transportní vrstvě.



Obrázek 2: Hlavička IP protokolu

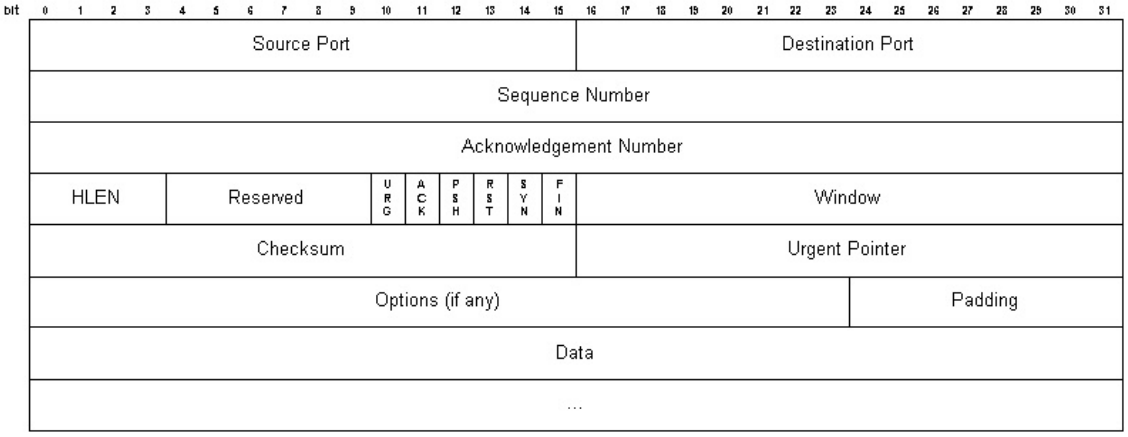
3.2.1 Funkce

TCP zajišťuje komunikační službu mezi aplikacemi a internetovým protokolem, tedy kdykoliv si aplikace přeje poslat data přes internet použitím IP, může software zažádat o TCP spojení a nechat TCP zacházet s detaily IP, namísto rozložení dat na menší části a zaslání řady menších IP žádostí. Z důvodů popsaných v kapitole 3.1.2, je nutnost zabránit ztrátě IP datagramů při cestě. TCP detekuje tyto problémy, vyžádá si znovuzaslání ztracených dat, přetřídí data, která přišla mimo pořadí a dokonce pomáhá minimalizovat zahlcení na síti. Jakmile TCP obdrží znovusestavenou sekvenci dat, přeposílá ji aplikacím. Proto je TCP abstrakcí komunikace aplikací se síťovými prostředky.

3.2.2 TCP segment

TCP přijímá data z aplikační vrstvy, rozdělí je na menší části a na konci procesu přidá TCP hlavičky. Tímto vytvoří tcp segment. TCP segment je následně zapouzdřen do IP datagramu. TCP segment se tedy skládá z hlavičky a vlastních dat. TCP hlavička obsahuje deset povinných polí a dále pole pro nepovinné rozšíření. Pole hlavičky jsou ukázány na obrázku 4. Povinné pole jsou následující:

- **zdrojový port** - 16bit zdrojový identifikátor
- **cílový port** - 16bit cílový identifikátor
- **pořadové číslo odesílaného bajtu** - sekvenční číslo, pořadové číslo prvního bajtu TCP-segmentu v toku dat (32 bitů), číslování začíná od náhodně zvoleného čísla



Obrázek 3: Hlavička TCP segmentu

- **pořadové číslo přijatého bajtu** - číslo následujícího bajtu, který je příjemce připraven přijmout, příjemce potvrzuje, že přijal vše do tohoto bajtu
- **délka záhlaví** - vyjadřuje se v násobcích 32 bitů
- **délka okna** - přírůstek pořadového čísla bajtu, který bude ještě příjemcem akceptován
- **Příznaky** - kontrolní bity sloužící k řízení:
  - **URG** - příznak naléhavých dat
  - **ACK** - příznak platného pořadového čísla přijatého bytu
  - **PSH** - příznak předání dat aplikaci
  - **RST** - reset spojení
  - **SYN** - odesílatel začíná s novou sekvencí číslování, TCP-segment nese pořadové číslo prvního odesílaného bajtu
  - **FIN** - konec přenosu, ukončení spojení
- **Kontrolní součet** - kontrolní součet segmentu, vypočítaný z přenášených dat a dat obsažených v hlavičce
- **Ukazatel naléhavých dat** - je-li nastaven URG příznak, pak toto pole ukazuje na poslední zaslaný urgentní byte, jehož pozice je počítána z pořadového čísla odesílaného bytu
- **rezervováno** - pole volných bitů pro budoucí využití

3.2.3 Operace protokolu

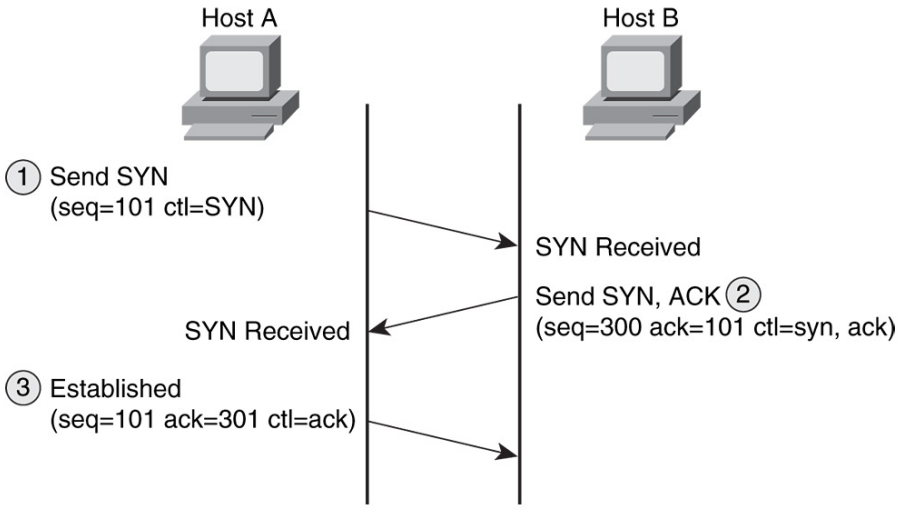
TCP protokol může být rozdělen do tří fází. Spojení se musí navázat několika krokovým procesem předtím, než se mohou zasílat vlastní data. Po zaslání dat (druhá fáze) se musí spojení uzavřít a následně se pak mohou uvolnit všechny prostředky svázané s přenosem (paměť, atd.). TCP spojení je navazované v operačních systémech pomocí socketu, programovatelného rozhraní, které reprezentuje lokální bod pro komunikaci. Během existence TCP spojení, toto rozhraní musí projít jednotlivými stavy:

- **listen** - reprezentuje čekání severu na žádost od klienta
- **syn-sent** - byla zaslána žádost na spojení od klienta
- **syn-recieved** - server přijal od klienta první TCP segment - segment s příznakem SYN
- **estabilished** - spojení navázáno, v tomto stavu mohou oba konce současně přenášet data (duplexní spoj)
- **fin-wait-1** - čekání na ukončení spojení, aktivní uzavření spojení
- **fin-wait-2** - čekání na ukončení spojení
- **close-wait** - přechod do pasivního uzavření spojení
- **last-ack** - druhá strana odeslala všechna data a signalizuje ukončení spojení segmentem
- **time-wait** - všechna data byla již oběma směry předána. Je nutné pouze potvrdit úplné uzavření spojení.
- **closed** - spojení ukončeno, uvolňují se hw prostředky

3.2.4 Zajištění spojení

K zajištění spojení využívá TCP proces zvaný Three-way handshake. Předtím, než se klient pokusí připojit k serveru, musí být server připraven a naslouchat na daném portu k otevření spojení (pasivní spojení). Jakmilie je navázáno pasivní spojení, klient může začít s aktivním otevřením. K zajištění spojení je potřeba provést tři kroky.

1. **SYN** Aktivní spojení je vyžádáno klientem požadavkem SYN na server. Klient nastaví sekvenční číslo na náhodnou hodnotu A.
2. **SYN-ACK** Na odpověd odešle server zprávu SYN-ACK. Pořadové číslo přijatého bytu (acknowledge number) na nastaveno o jedna větší (A+1) než hodnota čísla poslaného krokem SYN a sekvenční číslo je nastaveno na novou náhodnou hodnotu B.



Obrázek 4: Three way handshake

3. **ACK** Nakonec klient pošle ACK zpět na server. Sekvenční číslo je nastaveno na hodnotu A+1 a pořadové číslo přijatého bytu je nastaveno na hodnotu B+1.

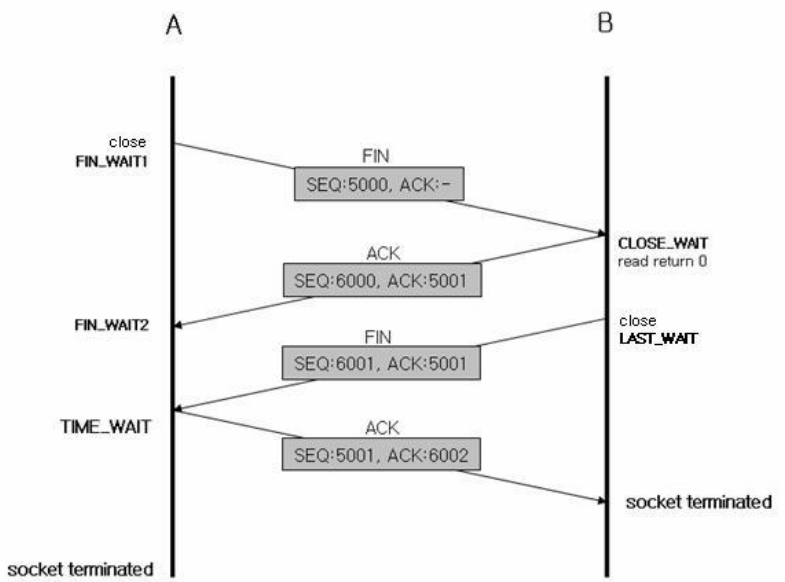
Od tohoto bodu si obě strany, tedy jak klient tak i server, vyměnili informace o spojení. Dále je zajištěna plně duplexní komunikace. Stav spojení je ve stavu estabilished.

3.2.5 Ukončení spojení

Ukončení spojení zpravidla zahajuje klient, ale spojení může ukončit jakákoliv strana. Spojení se ukončuje metodou zvanou Four-way handshake. Kdykoliv si jedna ze stran přeje ukončit spojení, zašle FIN zprávu druhé straně, která odpoví s ACK zprávou a následuje stejný postup akorát s prohozením stran. Strana, která zahájila ukončení FIN zprávou musí čekat, než ji přijde zpráva z druhé strany nebo vyprší hodnota nastavená v časovači. Po tuto dobu je port nedostupný pro nové spojení. Spojení může být napůl otevřené, tzn. že jedna strana ukončuje spojení, ale druhá nikoliv. Strana, která ho ukončuje již nemůže zasílat data, ale druhá strana může, proto ukončující strana by měla pokračovat ve čtení dat, dokud druhá strana neukončí spojení také. Celý proces ukončování spojení je zakreslen na obrázku 5.

3.3 Sockety

Socket (v oblasti počítačových sítí) je abstraktní rozhraní - koncový bod komunikačního toku v počítačové síti. Dnes je většina komunikace na síti založena na protokolu IP a proto je většina síťových socketů nazývána Internetovými sockety. API socketu je rozhraní, obvykle poskytováno operačním systémem, které umožní aplikacím používat a ovládat sockety. Dnešní API socketů je obvykle založené na standatdu BSD socketů. Adresa



Obrázek 5: Four way handshake

socketu je obvykle tvořena kombinací IP adresy a čísla portu, stejně jako je telefonní číslo tvořeno vlastním číslem a telefonní předvolbou.

3.3.1 Přehled

Internetový socket je charakterizován unikátní kombinací následujícího:

- Lokální adresou socketu: lokální IP adresa a číslo portu
- Adresou vzdáleného socketu: IP adresa, pouze pro TCP sockety, které jsou ve stavu ESTABILISHED
- Protokolem: protokol transportní vrstvy (TCP, UDP, . . .)

Uvnitř operačního systému nebo aplikace, která socket vytvořila, je socket identifikován unikátní hodnotou typu integer nazývanou socket identifier, socket number nebo socket deskriptor (framework Qt). Operační systém předává data příchozího IP packetu k příslušné aplikaci vyextrahováním informací k socketu z hlavičky a oddělením hlavičky od aplikačních dat.

3.3.2 Typy socketů

Internetové sockety mohou být rozděleny do několika typů:

- **Datagramové sockety** - také známé jako bezespojové sockety, které používají protokol UDP

- **Stream sockety** - známé jako spojové sockety, používající TCP protokol.
- **Raw sockety** - typicky dostupné ve směrovačích a jiném síťovém vybavení. u nich se obchází transportní vrstva a aplikace může přímo pracovat se surovým L2 rámcem.

Existují také jiné sockety než internetové, implementované na jiném transportním protokolu jako je např. System networks architecture (SNA).

3.3.3 Rozhraní BSD sockets

Rozhraní BSD sockets poskytuje aplikacím přístup k transportní a nižším vrstvám protokolového zásobníku. Poprvé se objevilo v operačním systému BSD Unix 4.1, který byl vyvinut na University of California, Berckley. Od té doby se stalo natolik oblíbeným a zažitým, že se stalo standardním aplikačním rozhraním k protokolům rodiny TCP/IP. Koncept tohoto rozhraní byl později přejat i jinými platformami, zejména ve světě PC. Rozhraní Sockets je navrženo, aby se jevilo jako rozšíření Unixového souborového systému. Tímto je možno pracovat se stejnými funkcemi jak pro práci se soubory, tak i pro síťovou komunikaci. Ale protože je síťová komunikace značně složitější, než práce se soubory, obsahuje rozhraní Sockets mnoho dalších podpůrných funkcí, které nejsou pro běžnou práci se soubory zapotřebí. Pro přístup k vrstvám protokolového stacku rozhraní Sockets vybrali autoři poměrně vysoký stupeň abstrakce. Je proto použitelné nejen pro protokoly rodiny TCP/IP, ale i pro protokoly zcela odlišné. Toho využili například tvůrci rozhraní Windows Sockets pro operační systém Windows NT, kde jsou mimo jiné podporovány protokoly IPX/SPX a je možné možné doplňování dalších rodin protokolů. Bohužel, je díky tomu nutno při volání jednotlivých funkcí rozhraní specifikovat větší počet parametrů.

3.3.4 Režimy práce socketů

Z hlediska funkce otevírání spojení rozlišujeme dva typy socketů - aktivní a pasivní. Jejich názvy odpovídají názvům použitým v RFC793 - způsob otevření TCP spojení. Pasivní socket je socket, který sleduje žádosti o navázání spojení a odpovídá na ně. Do sledovacího stavu se převádí funkcí listen(). Aktivní socket je tím, který žádost o navázání spojení iniciuje, což ve TCP protokolu představuje vyslání první SYN žádosti na adresu pasivního socketu. Každý socket se může nacházet v jednom ze dvou módů - blokujícím či neblokujícím. Tento mód pak následně ovlivňuje chování funkcí, které nad tímto socketem pracují. Jejich význam spočívá v tom, že ne všechny funkce rozhraní mohou být provedeny okamžitě a ihned pokračovat v běhu aplikace. Například běh funkce pro navázání spojení použit v síti s velmi vysokou odezvou, může trvat značně dlouhou dobu, v některých krajních případech až desítky sekund. Je to dáno proto, že navázání spojení je prakticky výměna několika packetů mezi iniciátorem a naslouchačem a v pomalých sítích tato výměna trvá nějaký čas. Nebo také naslouchající je současně zahlcen několika požadavky a nemůže odpovědět ihned. V některých případech toto nemísí být žádoucí a proto je možnost neblokujícího spojení. V praxi to pak znamená využití událostí, které nastanou, jakmile je některá z funkcí ukončena a aplikace na

to může zareagovat. Typické použití pro neblokující sockety jsou aplikace založeny na přístupu více navazovaných spojení.

4 Analýza

4.1 Požadavky

Jak již vyplývá ze zadání, je požadavkem na vypracování diplomové práce vytvořit novou verzi konzolového serveru, která by měla být jednodušší než stávající verze, ale na druhou stranu by měla mít možnost snáze doimplementovat nové funkcionality. Také bylo dohodnuto, že bude obsahovat jiný systém autentizace uživatele. Analýza autentizace je popsána v kapitole 4.2.

4.2 Autentizace uživatele

4.2.1 Původní systém autentizace

Původní systém autentizace byl závislý na webovém prostředí. Po přihlášení na úvodní stránce se vytvoří identifikátor relace (session ID), který dále identifikuje autentizované uživatele. Konzolový server tedy neověřuje samotné uživatele, ale pouze existenci tohoto identifikátoru. Na tento identifikátor dále navazují další ověřovací mechanismy, jako je ověření, zda je připojované zařízení v rezervaci uživatele.

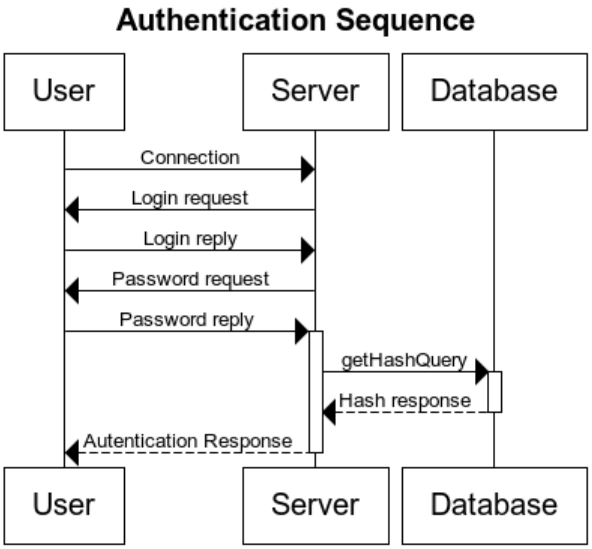
4.2.2 Navrhovaný systém autentizace

Hlavní výhodou nového systému autentizace by měla být nezávislost na webovém prostředí. Přihlašovaní bude probíhat na vyžádání ze strany serveru a výměnou přihlašovacích údajů v textové formě. Ověření zadaného hesla bude vyžadovat připojení k databázi, která obsahuje uživatelská data. Pro tuto databázi byla vybrána databáze, kterou používá webový server. Více o databázi je popsáno v kapitole 4.3. Diagram znázorňující celý průběh ověření hesla je na obrázku 6.

4.3 Databáze

Tabulky obsažené v databázi virtlabu jsou následující:

- **files** - soubory používé v úlohách a rezervacích, obrázky topologií, ukázky konfigurací nebo popis zadání
- **languages** - seznam jazyků
- **localization** - překlad názvů v databázi
- **mail** - komunikace mezi uživateli v systému
- **motd** - tzv. Zprávy dne (Message of the Day)
- **reservation_remote_users** - spolupracovníci z různých lokalit
- **reservation_users** - spolupracovníci na rezervaci



Obrázek 6: Postup ověření hesla

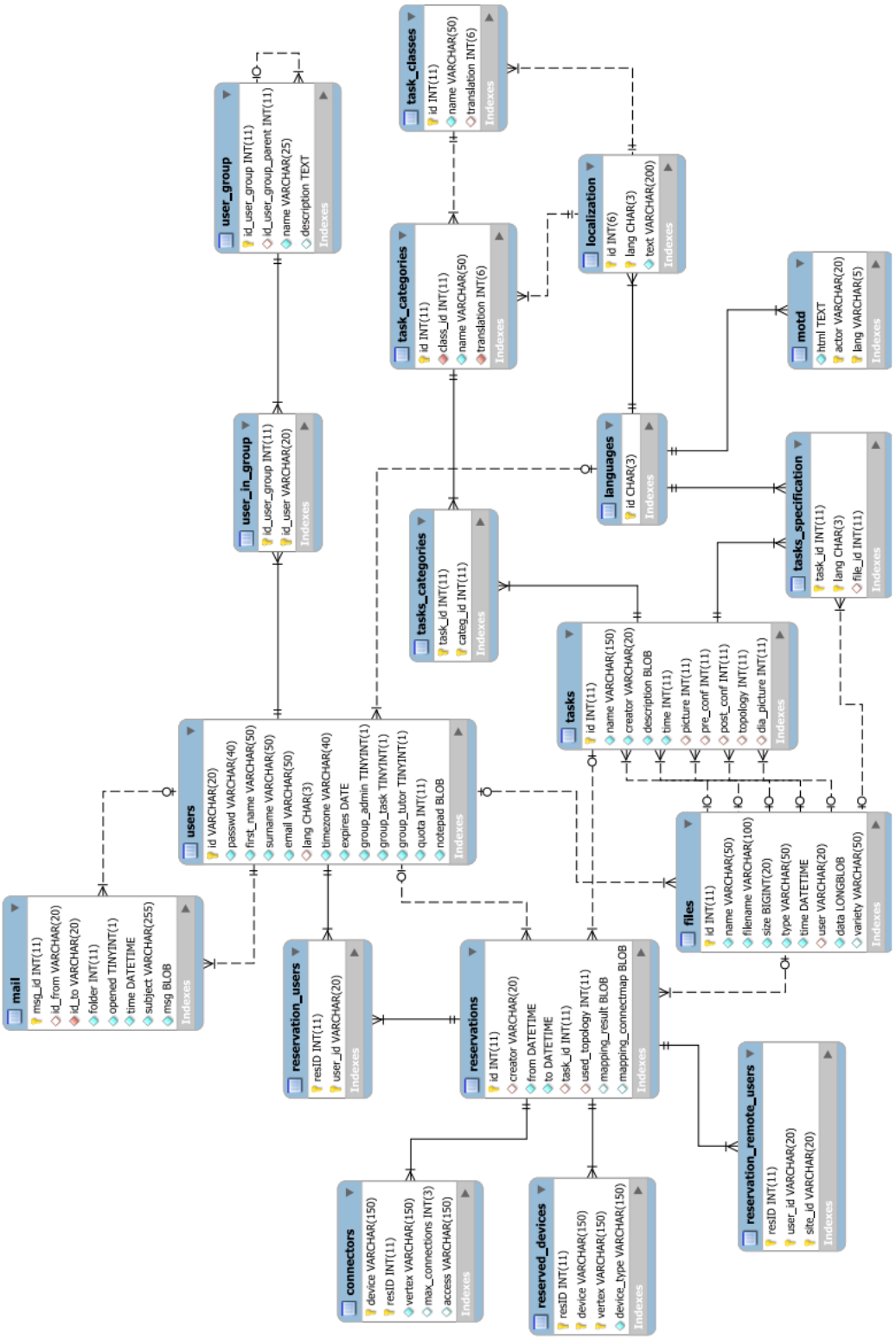
- **reservations** - rezervace na daný čas
- **reserved_devices** - zařízení potřebná pro rezervace
- **task_categories** - kategorie pro úlohy
- **task_classes** - třídy úloh
- **tasks** - úlohy
- **tasks_categories** - přiřazení úloh do kategorií
- **tasks_specifikations** - specifikace úloh dle jazyka
- **user_group** - uživatelské skupiny
- **user_in_group** - přiřazení uživatelů do skupin
- **users** - uživatelé, práva a uživatelská data

ER diagram popisující databázi virtlabu je vidět na obrázku 7.

4.3.1 Tabulka users

Tato tabulka obsahuje informace o všech uživatelích, kteří pracují s virtlabem. Nejdůležitější sloupce této tabulky jsou:

- **id** - identifikátor uživatele, pod kterým se přihlašuje do systému. V lokalitě VŠB je to standartně třípísmenná zkratka odvozená z přijimení plus dvě až tři číslice udávající pořadí číslic v systému (např. ABC123).



Obrázek 7: Databáze virtlabu

- **passwd** - Může obsahovat dvojí informace a to:
  - otisk hesla (hash) uživatele vytvořený algorytmem SHA-1
  - řetězec '–LDAP–' značící, že heslo se ověřuje na skolních LDAP serverech.
- **first_name** - jméno uživatele
- **surname** - přijímení uživatele
- **group_admin** - bool hodnota značící, zda je uživatel administrátorem.

4.3.2 Tabulky rezervací

Pro zjištění aktuálně dostupných rezervovaných zařízení a umožnění připojení k těmto zařízením jsou nutné tyto tabulky:

- **users** - popsána v kap. 4.3.1. obsahuje id uživatele.
- **reservation_users** - vazební tabulka umožňující mít pro jednoho uživatele více rezervací a k rezervaci připojit více uživatelů (spolupracovníků).
- **reservations** - stěžejní tabulka rezervací, obsahující identifikátor, id uživatele, který rezervaci vytvořil a hlavně čas platnosti dané rezervace. Proti tomuto času se bude muset ověřovat aktuální čas na konzolovém serveru, aby mohl vypsat aktuální seznam rezervovaných zařízení.
- **reserved_devices** - tabulka zařízení pro jednotlivé rezervace. Opět více zařízení může pripadat na jednu rezervaci a vice versa.

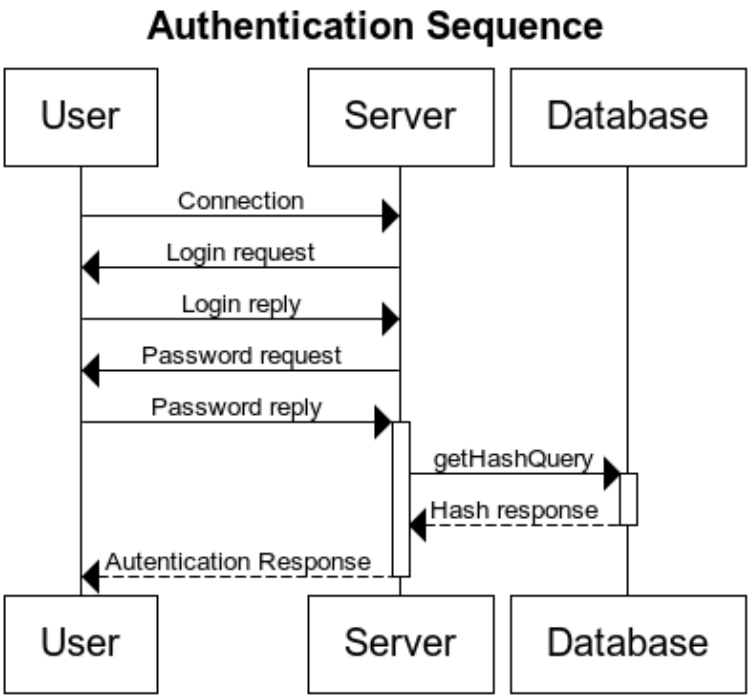
Samotná tabulka reserved_devices je tvořena následujícími sloupci:

- **resID** - cizí klíč na tabulku reservations
- **device** - název zařízení, které je použito ve schématu zapojení konkrétní topologie
- **vertex** - unikátní název zařízení, který je ve formě idzarizeni@lokalita. Pomocí tohoto identifikátoru se vybírá, na které zařízení se uživatel bude připojovat.
- **device_type** - typ zařízení (router, switch, pc...)

4.4 Vzdálená lokalita

4.4.1 Popis

Jak již bylo popsáno v kapitole 2 systém Virtlab je distrubuovaný, čili existuje více lokalit se svými prvky, databází a vebovým serverem. Také každá lokalita obsahuje svou vlastní instanci konzolového serveru. A jelikož mají být dostupné prvky i mezi lokalitami, je nutné mezi těmito konzolovými servery zajistit prostředky pro komunikace, autentizaci a přenos dat. Situaci znázorňuje schéma na obr. 8



Obrázek 8: Autentikace uživatele

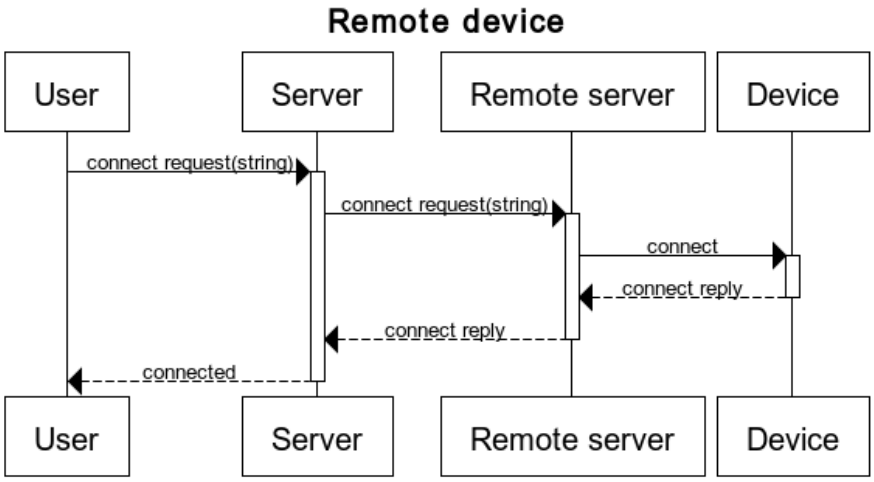
4.4.2 Tunelování mezi lokalitami

Pro zajištění propojení mezi různými VLAN v jednotlivých instancích virtlabu a také propojení do jednoho ethernetového segmentu je použita komponenta tunelovacího serveru.

Hardwarové zařízení, na kterém běží instance tunelovacího serveru, musí obsahovat dvě rozhraní typu ethernet. Jedno z nich je připojeno k síti Internet a druhé je spojeno rozhraním trunk (802.1q) k propojovacímu prvku VLMUX. V době, kdy je aktivována rezervace, se nakonfiguruje tabulka (tabulka přesměrování), ve které jsou uvedeny vždy: VLAN ID rámce z lokální sítě, VLAN ID sítě, do které bude rámec zaslán a IP adresa vzdáleného či lokálního tunelovacího serveru. Tunelovací server na straně, na které je zapojena trunk linka, zachycuje ethernetové rámce (s VLAN záznamem) a kontroluje, zda jsou obsaženy v tabulce přesměrování a podle ní je přetaguje a pošle zpět s jinou hodnotou VLAN ID nebo přes Internet na danou IP adresu tunelovacího serveru. Tímto způsobem dojde k propojení (virtuální) ethernetu skrze ethernetový tunel.

4.4.3 Komunikace konzolového serveru mezi lokalitami

Kdykoli se uživatel bude chtít připojit k zařízení na jiné lokalitě, musí komunikace probíhat přes dva konzolové servery. Sekvenční diagram popisující tuto komunikaci je zobrazen na obrázku 9. Samotná komunikace probíhá v těchto krocích: Prvně je zaslán konzolovému serveru požadavek na spojení se zařízením. Jako argument je poslán unikátní



Obrázek 9: Připojení k vzdálenému konzolovému serveru

identifikátor zařízení. Server si z konfiguračních dat zjistí, zda je zařízení lokální či vzdálené(tzn. umístěno v lokalitě serveru či je v jiné lokalitě). Dále vytvoří nový socket a spojí se na IP adrese a portu takovém, jaké vyčetl z konfiguračního souboru. Je-li zařízení vzdálené, nastaví se pevný port komunikace, mezi vzdálenými servery. Při lokálním zařízení již pak normálně probíhá komunikace. Spojení na server se nastaví do režimu přeposílání dat a již se hlídá jen výskyt kontrolního znaku ukončení komunikace se zařízením. Při vzdáleném zařízení se vytvoří nové spojení na pevném portu, který značí, že se jedná přesně o tento typ komunikace (spojení dvou konzolových serverů). Po vytvoření spojení vzdálený server čeká na přijetí unikátního identifikátoru zařízení. Po přijetí tohoto řetězce je na vzdáleném serveru vytvořeno spojení se zařízením dle informací, nacházející se ve vlastním konfiguračním souboru. Po vytvoření tohoto spojení pošle potvrzující informaci zpět původnímu serveru, který informuje uživatele a nastaví se do režimu přeposílání dat.

Samotná autentizace navázání spojení mezi instancemi konzolového serveru je na základě dvou údajů. Jedním z nich je seznam známých konzolových serverů a jejich IP adres u každé instance. Druhým je identifikace příslušného portu na kterém konzolové servery mezi sebou komunikují (je unikátní).

4.5 Přístupy

4.5.1 Běžný uživatel

U běžného uživatele je představa běžné práce se zařízeními. Hlavní službou poskytovanou uživateli je připojení k zařízení a to buď v rámci dané lokality nebo v v lokalitě jiné. Chování serveru pro připojení k zařízení v lokalitě jiné než je lokalita uživatele je samozřejmě odlišné. Více je popsáno v kapitole 4.4.3. Mimo připojení k zařízení jsou

příkaz	parametr	popis
connect	devID	připojení k danému zařízení
exit	-	ukončí připojení ke konzolovému serveru
help	-	zobrazí nápovědu s výpisem všech dostupných příkazů
list	-	připojí se k databázi a zjistí seznam všech zařízení, které jsou dostupné v právě probíhající rezervaci
ostatní	-	vypíše chybovou hlášku - nerozpoznaný příkaz

Tabulka 1: Tabulka příkazů administrátora

příkaz	parametr	popis
connectionlist	-	zobrazí všechna spojení od uživatelů
terminate	connectionID	terminuje spojení s unikátním connectionID

Tabulka 2: Tabulka příkazů administrátora

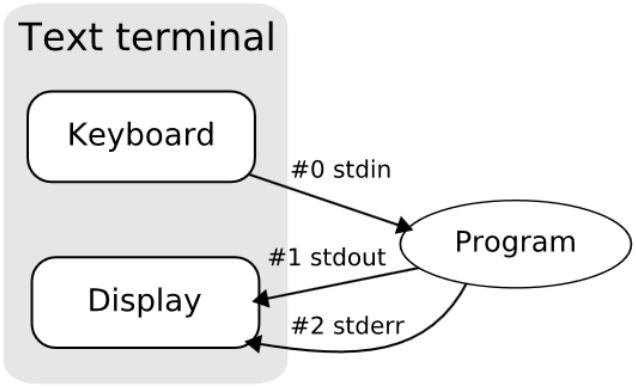
na konzolovém serveru dostupné i jiné příkazy. Kompletní seznam příkazů je popsán v tabulce 1.

4.5.2 Administrátor

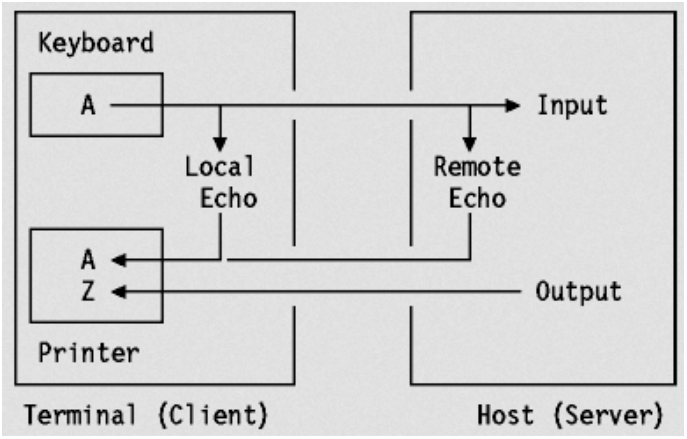
Narozdíl od běžného uživatele by měl administrátor mít k dispozici i příkazy pro ovládání všech spojení, která jsou na konzolovém serveru k dispozici. Konkrétně by mělo jít o příkazy, která zobrazí současné spojení od uživatele (socket s připojením ke vzdálenému serveru nebo socket spojení se zařízením se ve výpisu neukáže) a možnosti terminace tohoto spojení. Každé spojení by tedy mělo mít jednoznačný identifikátor, která se v jednom výpisu zobrazí a na druhou stranu i použít jako argument pro terminaci spojení. Při terminaci se automaticky odpojí veškeré ostatní sockety. Administrátor se bude moci identifikovat přihlášením přes zvláštní port, nastavený v konfiguračním režimu nebo by měla být aplikace konzolového serveru schopna ověřit uživatelský záznam v databázi (sloupec group_admin popsaný v kapitole 4.3.1). Výpis unikátních příkazů pro administrátora je vypsán v tabulce 2.

4.6 Zpětná vazba komunikace

Jakákoliv komunikace uživatele s jiným prvkem musí obsahovat zpětnou vazbu. Tato zpětná vazba zobrazuje práci uživatele s prvkem a reaguje na uživatelovy akce. Je to tedy prezentace vstupu na nějaký výstup, obvykle obrazovku. Příkladem je práce v terminálu pro komunikaci s počítačem. Uživatel zadávající příkazy na klávesnici očekává zobrazení tohoto vstupu na obrazovce počítače, se kterou terminál pracuje. Obrázek ukazující komunikaci je na obrázku 10. Existují dva typy zpětné vazby při komunikaci se zařízením - lokální a vzdálená. Lokální zpětná vazba (local echo) je případem zpětné



Obrázek 10: Komunikace se zpětnou vazbou



Obrázek 11: Lokální a vzdálená zpětná vazba

vazby, kdy zadávané znaky posílá k zobrazení strana, která znaky zadává. Tedy máme-li, nějakou aplikaci, která vyčítá text a následně ho posílá ke zpracování, tak aplikace zobrazuje text ještě před zasláním. Narozdíl od toho vzdálená zpětná vazba (remote echo) je typem zpětné vazby, kde vlastní text k zobrazení posílá strana, ke které se navazuje komunikace. Případem jsou zařízení firmy Cisco připojené na konzolovém portu. Každý znak zadaný na straně uživatele je poslán k zařízení, které ho zpracuje a předá textový výstup zpátky ke stanici uživatele, která ho teprve poté zobrazí. S tímto je nutné počítat při návrhu aplikací pro komunikaci s těmito zařízeními. Jak vzdálená tak lokální zpětná vazba jsou zakreslené na obrázku 11.

4.7 Vícenásobné přístupy k zařízením

Každá rezervace prvků obsahuje seznam uživatelů, kteří se k těmto prvkům mohou připojovat, tak seznam samotných prvků. Z tohoto důvodu je tedy jasné, že se dostáváme

k tomu, že na jeden prvek se tedy prakticky může připojit více uživatelů. Existují možné způsoby jak tohle řešit a ty jsou následující:

- Omezit přístup k zařízení pouze na jednoho uživatele.
- Umožnit přístup k zařízení všem uživatelům, kde pouze jeden bude moct zasílat příkazy na zařízení.
- Povolit přístup k zařízení všem uživatelům bez omezení.

Poslední možnost byla ihned vyloučena a dále se uvažovalo o prvních dvou možnostech. První možnost je velice jednoduchá na implementaci, ale přesto byla nakonec vybrána možnost druhá s ohledy na spolupracující uživatele. Při implementaci bude nutné počítat tedy s následujícím:

- Pouze jeden socket na odchozí spojení. Bude tedy nutné kontrolovat stav socketu v jakém se nachází před spojením a pokud už je spojení vytvořeno, pak se na něj pouze napojit. V opačném případě musí aplikace navázat spojení se zařízením.
- Umožnit zápis pouze prvnímu uživateli. Příchozí data ostatních uživatelů budou ignorovány.
- Udržovat frontu všech uživatelů, kteří se chtějí připojit k zařízení. Důvod je prostý. Odpojí-li se první připojený uživatel, některý z ostatních uživatelů by měl převzít jeho práva zápisu na zařízení. Tedy při odhlášení uživatele s právem zápisu dojde k tomu, že uživatel bude odebrán z fronty a vyšle se informace uživateli, který je na vrcholu. Ten následně nastaví práva zápisu na kladnou hodnotu. Sekvenční diagram je zakreslen na obrázku .

4.8 Klientská aplikace

V novém návrhu konzolového serveru pro virtlab bylo počítáno i s tím, že bude k dispozici přehledná klientská aplikace s GUI rozhraním. Toto GUI by mělo být pro uživatele co nejsnadnější pro obsluhu a mělo by poskytovat jednoduchou možnost ovládání prvků. Samotná implementace by měla co nejvíce odpovídat návrhu aplikace, který obsahuje jednoduché menu s nástrojovou lištou a systém záložek reprezentující komunikaci mezi uživatelem a zařízením..

5 Implementace

5.1 Výběr frameworku

Vzhledem k tomu, že klientská aplikace obsahuje grafické rozhraní, je nutné vybrat sadu knihoven pro práci s tímto prostředí. Požadavkem na klientskou aplikace je také multiplatformnost, tzn. možnost spouštět aplikaci na různých operačních systémech, abychom neomezovali uživatele. Hlavní platformy dneška jsou:

- **Microsoft Windows**
- **Linux** - linuxové distribuce, dnes nejznámější asi Ubuntu
- **Mac OS X**

Pro každou tuto platformu bude vzhledem k zadání vytvořena verze klientské aplikace. Aby bylo možné vytvořit aplikaci na více než jedné platformě, můžeme postupovat dvěma způsoby:

- Vytvořit aplikaci pro každou platformu zvlášť s přihlédnutím ke specifikám dané platformy
- Použít sadu multiplatformních knihoven

Jelikož je vytvoření aplikace na každé platformě časově náročné a vyžaduje podrobnou znalost každého prostření, bylo přistoupeno ke kroku vybrat ke tvorbě aplikace některé ze sady multiplatformních knihoven. Takovýchto knihoven existuje celá řada a proto vypíši jen ty nejznámější. Jsou to:

- **GTK+** - původně vznikla pro potřeby grafického rastrového editoru GIMP a poté byla použita pro prostředí GNOME (Ubuntu)
- **Qt** - jedna ze dvou nejpopulárnějších knihoven pro vytváření aplikací s GUI. Vyvinut společností Trolltech, která jej prodala společnosti Nokia.
- **Swing** - velice populární knihovna fungující na platformě Java. Vyvinut firmou Sun a po odkoupení ve vývoji společností Oracle.
- **wxWidgets** - jedna z nejstarších knihoven

Vzhledem ke znalostem nativních jazyků, které knihovny využívají a vzhledem k jejich rozšířenosti a následné podpory, bylo vybíráno ze dvou knihoven - Qt a Swing. Každá z knihoven má své výhody a nevýhody.

Swing:

- Výhody
  - Dostupná s jakoukoliv Java instalací. Nepotřebuje žádné dodatečné knihovny připojené k programům.

  - Vlastní prvky psané v Javě.

- Nevýhody
  - Swing je zastaralý. Poslední aktualizace jádra byla již před slouhou dobou nové aktualizace se nechystají.
  - Obtížný na pochopení, obzvláště pro programátory UI na operačním systému Microsoft Windows.
  - Nedostatečně přizpůsobitelné prvky obsažené v jádru frameworku.
  - Pro dobrý vzhled aplikací jsou potřeba extra knihovny.

Qt:

- Výhody
  - Komplexní sada přizpůsobitelných prvků
  - Snadný na naučení a použití
  - Dobrá dokumentace
  - Dobrá podpora
  - Stále probíhající vývoj.
  - Používá nativní prvky a přiřazuje je do běžných API přes všechny platformy.
- Nevýhody
  - S aplikací je potřeba dodat i moduly, které jsou potřebné pro běh.

S přihlédnutím k výchozímu programovacímu jazyku, které každá z knihoven používá, C++ pro Qt a Java pro Swing, a vzhledem k převážení výhod bylo pro použití aplikace vybrána sada knihoven Qt.

Komponenta konzolového serveru neobsahuje žádné GUI prvky (jedná se o službu), ale přesto bude používat knihovny ze sady produktu Qt. Důvodem je to, že Qt neobsahuje jen sady pro práci s grafickým prostředím, ale i sady knihoven pro práci se síťovou komunikací, knihovny pro práci s databázemi a další. Jde o komplexní produkt, který je blíže popsán v kapitole 5.2.

5.2 Qt framework

Qt je multiplatformí aplikační framework, který je široce využit u aplikací, které využívají grafické prostředí (kdy je pak Qt označováno jako tzv. widget toolkit). Také je využíván u aplikací bez grafického prostředí jako jsou nástroje pro příkazovou řádku a konzolové servery. Qt standardně využívá jako programovací jazyk C++, ale navíc široce používá speciální generátor kódu (zvaný Meta Object Compiler nebo moc) spolu s několika makry k obohacení jazyka. Qt může být použito i s jinými programovacími jazyky díky použití vlastních zkompilovaných kódu. Běží na všech hlavních desktopových platformách a i

na některých mobilních. Obsahuje podporu pro rozšířenou možnost překladu aplikací, přístupy do SQL databází, XML parsování, podporu správy vláken, síťovou podporu a sjednocené multiplatformní API pro práci se soubory.

Qt je dostupno pod komerční licencí, GPL v3 a LGPL v2 licencí. Všechny verze podporují mnoho kompilátorů včetně GCC C++ kompilátoru a i kompilátoru Visual Studia.

Qt je v současnosti vyvíjen spoječností Digia, která vlastní technologie a obchodní značku. Qt Project je další strana vývoje, sjednocující nezávislé vývojáře a společnosti, které se snaží o vylepšení Qt. Před spuštěním Qt Project byl Qt vyvíjen oddělením pro vývoj Qt společností Nokia, která toto oddělení vytvořila po odkoupení norské společnosti Trolltech, která je původním tvůrcem Qt.

V současnosti je nejnovější verzí verze 5.1 vydaná v dubnu roku 2013. Verze nově obsahuje experimentální moduly pro vývoj na mobilních platformách Google Android a Apple iOS. Celkově se však neustále pracuje na vylepšování výkonu a stability všech prvků, které framework poskytuje.

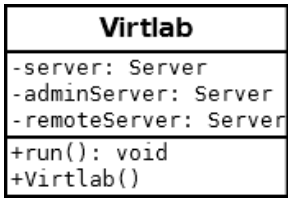
Inovace Qt od doby prvního vydání spočívá v několika klíčových konceptech:

- **Nástroje**
  - Qt Creator jako multiplatformní IDE pro C++ a QML (programovací jazyk založen na javascriptu)
  - qmake - nástroj, který generuje Makefile pro vývoj na různých platformách
- **Užití nativních vzhledů API** - Qt se snaží emulovat vzhled na plánované platformě, což občas vede k nesrovnalostem vzhledu. Na některých platformách (například KDE nebo MeeGo) je Qt nativním API.
- **Kompilátor metaojektů** - interpretuje různá makra z C++ kódu jako anotace a používá je ke generování přidaných C++ kódů s meta informacemi o třídách, použitých v programu. Tyto meta informace jsou pak použity k poskytnutí vlastností, které nejsou přímo v C++. Jsou to například signál/slot systém, nahlédnutí do objektů a asynchronní volání funkcí.
- **Napojení** - Qt má širokou škálu jazyků, se kterými jde použít. Pro některé jazyky však není poskytována plná podpora všech modulů. Mezi podporovanými jazyky jsou například: C#, Java, Pascal, Perl, PHP, Python, Ruby a další.

5.3 Implementace Konzolového serveru

Jak bylo zmíněno v kapitole 5.1 k implementaci konzolového serveru je použito frameworku Qt od společnosti Digia. Jako programovací jazyk byl vybrán jazyk C++, který je objektově orientovaný a k tomu se samozřejmě váže i rozdělení do tříd. V samotné aplikaci jsou tyto třídy:

- **Virtlab**
- **Server**



Obrázek 12: Třída Virtlab

- **ClientSocket**
- **Config**
- **Communication**
- **CCommand**
- **Database**
- **ConnectionManager**
- **LocalOutput**

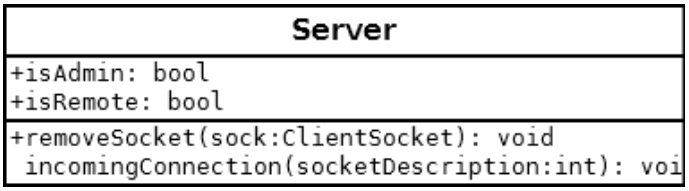
Mimo tyto třídy existují v aplikaci i třídy další, ale ty se vážou k třídám zde vypsaným a budou popsány v konkrétních kapitolách.

5.3.1 Třída Virtlab

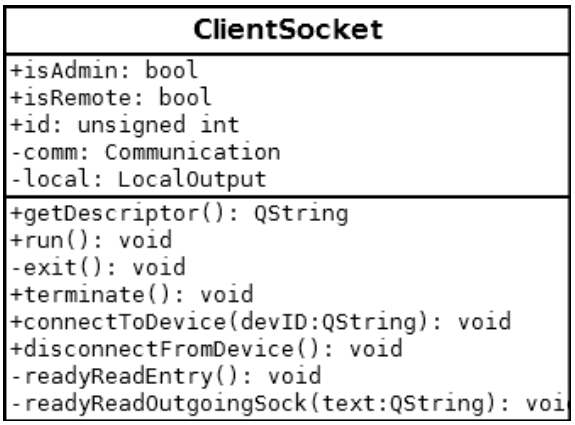
Třída Virtlab je jakousi spouštěcí třídou celé aplikace. Kromě konstruktoru obsahuje jedinou metodu a tou je metoda run, která nedělá nic jiného než že spustí naslouchání serverů na daných portech. Jak je vidět na obrázku 12 tak třída mimo to obsahuje i tři instance třídy Server. Tyto instance reprezentují tři porty, na kterých aplikace naslouchá. Jsou to:

- Port, na kterém iniciuje komunikaci běžný uživatel
- Port, na kterém iniciuje komunikaci administrátor.
- Port, na kterém probíhá komunikace mezi jednotlivými lokalitami.

Jakákoliv chyba, která vznikne ve chvíli, kdy by měl server začít naslouchat se automaticky ukládá do logu. Není-li v nastavení specifikována výchozí IP adresa, automaticky se vybere adresa z dostupných zařízení. Pokud ani tu nenalezne, pak je jako výchozí adresa použita IPv4 adresa localhostu (127.0.0.1).



Obrázek 13: Třída Server



Obrázek 14: Třída ClientSocket

5.3.2 Třída Server

Tato třída dědí ze třídy QTcpServer, která je jednou ze tříd sady QNetwork. Z této třídy reimplementuje metodu incomingConnection (obrázek 13), která má jako parametr identifikátor socketu (popsáno v kapitole 3.3.1). Díky tomuto deskriptoru vytvoří novou instanci třídy ClientSocket a předá ji správci spojení. Dále ještě naváže metodu removeSocket na signál této instance, který se vyvolá v případě, že došlo k odpojení navázaného spojení (jak nuceně tak ze strany uživatele).

5.3.3 Třída ClientSocket

Pravděpodobně nejdůležitější třída reprezentující spojení buď od uživatele nebo od jiného konzolového serveru. Podobně jako třída Server dědí ze třídy QTcpServer, tak třída ClientSocket dědí ze třídy QTcpSocket. Má několik metod, díky kterým může probíhat komunikace s uživatelem, ale vlastní ověřování probíhá ve třídě Communication. Komunikace s druhou stranou, tj. vlastní zařízení či jiný konzolový server, je reprezentován instancí třídy LocalOutput, kterou přebírá od třídy Config v metodě connectToDevice. Výběr probíhá podle parametru, kterým je unikátní identifikátor zařízení. Ukončení spojení s uživatelem v této třídě je možné třemi způsoby - klasické ukončení, které zahajuje uživatel, vynucené ukončení, které si může vyžádat administrátor nebo vynucené ukončení, které se děje automaticky (ztráta spojení, neaktivita).

5.3.4 Třída Communication

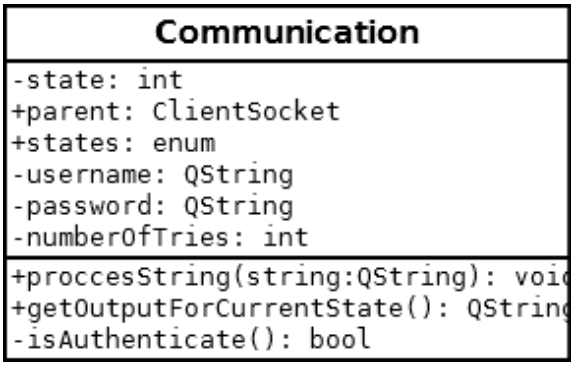
Třída Communication je částí, která definuje a zpracovává akce uživatele. Jak je vidět na obrázku 15 obsahuje i vlastní uživatelské data (uživatelské jméno a heslo). Tyto data si postupně ukládá a následně v metodě isAuthenticate vytvoří ze zadaného hesla SHA-1 hash, dotáže se přes mezivrstvu na databázi o hash uloženého hesla a porovná jejich shodu. Jsou-li shodné tak je uživatel autentizován. Není-li shoda, tak se přistupuje k výzvě zadání nového hesla. Dále třída obsahuje několik stavů:

- **UnLogged** - úvodní stav, který je iniciován v konstruktoru třídy ClientSocket. Značí, že uživatel se právě připojil k serveru.
- **Logged** - stav, ve kterém uživatel zadal přihlašovací jméno a čeká se na zadání hesla
- **SuccessAuth** - jméno a heslo bylo zadáno a autorizace proběhla úspěšně. Uživateli se vrátí text značící úspěšné přihlášení.
- **FailedAuth** - autorizace se nezdařila a běží počitadlo pokusů na úspěšné zadání hesla
- **CommandEntered** - stav, ve kterém se zadávají příkazy jak pro normálního uživatele tak i pro administrátora.
- **Connecting** - stav, ve kterém se nachází třída jen pokud se uživatel přihlašuje ke vzdálenému zařízení. Značí, že spojení se serverem bylo navázáno, ale čeká se na odpověď na navázání na konkrétní zařízení.
- **Connected** - bylo navázáno spojení se zařízením a může probíhat komunikace. Cokoliv je přijato ke čtení, tak se odešle na výstupní zařízení.
- **Exit** - spojení s uživatelem je ukončeno. Neprobíhá komunikace a uvolňují se prostředky.

Tyto stavy popisují současné chování odpovědí serveru na reakce od uživatele. Nastavují se buď v třídě Communication nebo v jeho rodiči, třídě ClientSocket (inicializace nebo stavy pro spojení s odchozími zařízeními).

5.3.5 Třída CCommand

Jedná se o návrhový vzor Abstract Factory, která se využívá tehdy, když máme skupinu objektů se společným tématem, v tomto případě každá definuje jeden příkaz zadaný na konzoli. Prakticky vše funguje tak, že ve třídě CCommand je definovaná metoda create, která slouží k vytvoření konkrétní instance třídy, kterou je příkaz reprezentován. Do této metody je poslán řetězec, který se rozdělí a první část se otestuje. Je-li shoda s některým s příkazů, vytvoří se jeho instance a vrátí se jako návratová hodnota metody. Execute je druhou metodou, tentokrát virtuální a je implementována v konkrétních reprezentacích. Ty všechny implementují ze třídy CCommand. Jejich seznam je následující:



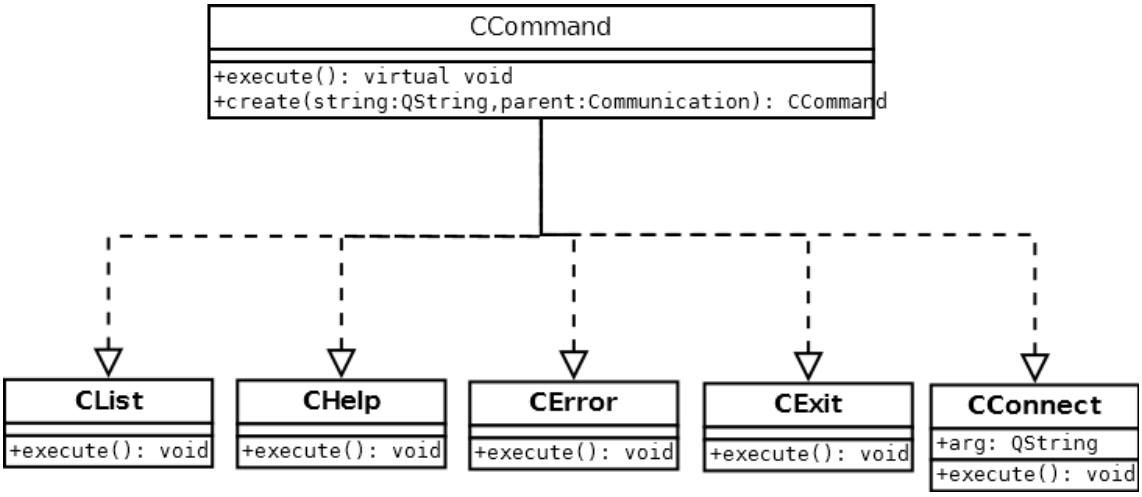
Obrázek 15: Třída Communication

- CList - zobrazí seznam všech dostupných zařízení, konkrétně zavolá metodu getDeviceList, které předá uživatelské jméno rodiče(ukazatel na třídu Communication, který instanci vytvořilo).
- CHelp - vypíše seznam všech použitých příkazů.
- CError - vypíše chybovou hlášku (”Příkaz nebyl rozeznán!”)
- CExit - nastaví stav rodiče na Exit a vyvolá ukončení spojení
- CConnect - zavolá metodu connectToDevice a pošle ji argument příkazu
- CConnList - ze třídy správy spojení si vyžádá seznam spojení a vypíše jej.
- CConnTerminate - pošle argument příkazu (unikátní číslo spojení) do instance třídy ConnectionManager a tím vyvolá ukončení spojení

Přidání nového příkazu se skládá ze dvou kroků: vytvoření nové třídy, která bude dědit ze třídy CCommand a tím i implementace metody execute. Druhým krokem je přidání rozeznání příkazu do metody Create, která je rozdělena na dvě části a to uživatelská a administrátorská (porovnání příznaku isAdmin u rodiče). Pokud bude shoda tak vrátí instanci nově vytvořené třídy. Třída CCommnad a dědičnost navazujících tříd je zakreslena na diagramu na obrázku 16.

5.3.6 Třída Config

Jedná se o sigleton, který reprezentuje konfigurační hodnoty v aplikaci. Singleton je název pro jeden z návrhových vzorů. Využívá se tehdy, kdy je potřeba, aby v aplikaci byla jen jedna instance dané třídy. Z toho vznikl i samotný název tohoto návrhového vzoru (singleton = jedináček). Samotná instance je tedy obsažena v této třídě jako statická privátní proměnná. Dá se k ní samozřejmě přistupovat, ale pouze přes metodu getInstance, která pokud je instance prázdná (hodnota NULL) zavolá konstruktor třídy. Tento konstruktor je podle návrhového vzoru také privátní. Volba návrhového vzoru sigleton vyšla z toho,



Obrázek 16: Třída CCommand

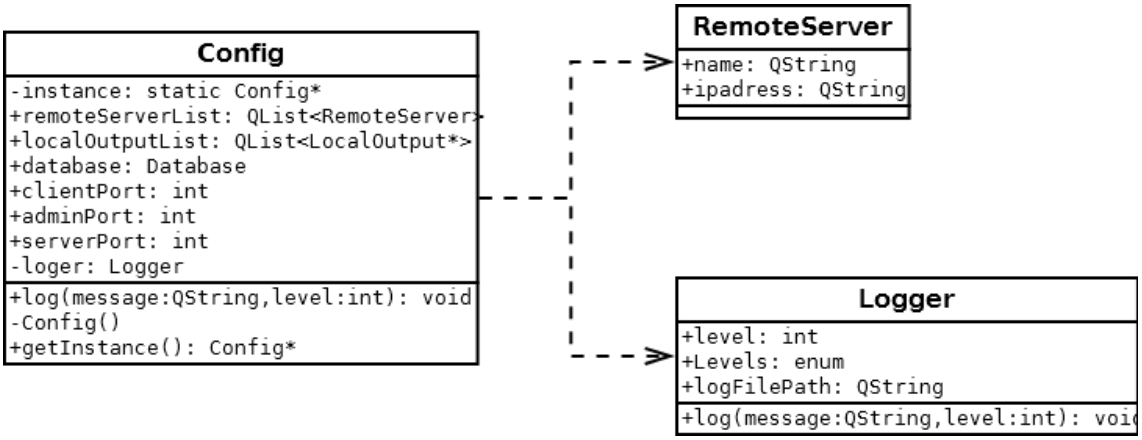
že konfigurace je uložena v textových souborech a jejich vyčtení probíhá v konstruktoru. Bylo by tedy značně neefektivní buď předávat pokaždé ukazatel na jednu instanci nebo pokaždé při vytvoření nové instance znova vyčítat informace ze souboru. Samotná konfigurace se dá rozdělit do čtyř hlavních bodů:

- Konfigurace aplikace - soubor config.conf
- Konfigurace databáze - soubor databaze.conf
- Seznam vzdálených konzolových serverů - soubor conf-servers.conf
- Seznam zařízení - conf-devices.conf

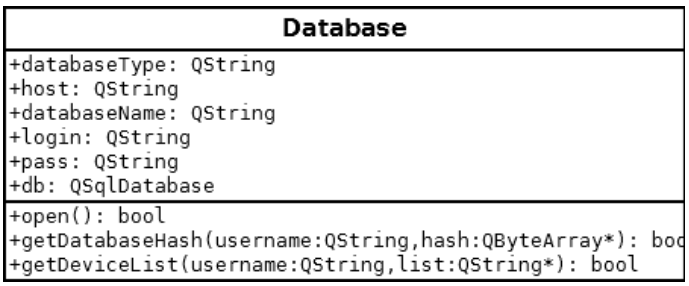
První část, konfigurace aplikace se skládá z nastavení portů aplikace. Je to port pro přístup standardního uživatele, port pro přístup uživatele s administrátorskými právy a nakonec port pro komunikaci mezi jednotlivými konzolovými servery. Poslední částí je nastavení loggovací úrovně, podle které se bude zapisovat do log souboru.

Druhou částí je soubor s konfigurací pro přístup k databázi virtlabu. Obsahuje přístupová data (uživatelské jméno a heslo), dále řetězec s názvem vlastní databáze a typ použité databáze. Tento typ se musí shodovat s řetězci, používanými frameworkem Qt pro práci s databází. V současné době používá virtlab databázi MySql a řetězec pro identifikaci této databáze je MYSQL.

Další částí je seznam vzdálených serverů. Tento seznam slouží jen pro případ, kdy se uživatel připojuje k zařízení, které se nachází v jiné lokalitě. V tom případě vybere řetězec z názvu zařízení identifikující lokalitu a podle ní si vyhledá ip adresu této lokality. Pro servery byla vytvořena speciální třída RemoteServer, která uchovává jméno a ip adresu lokality. Všechny instance této třídy jsou pak uchovávány v kolekci typu QList s názevem remoteServerList v třídě Config.



Obrázek 17: Třída Config



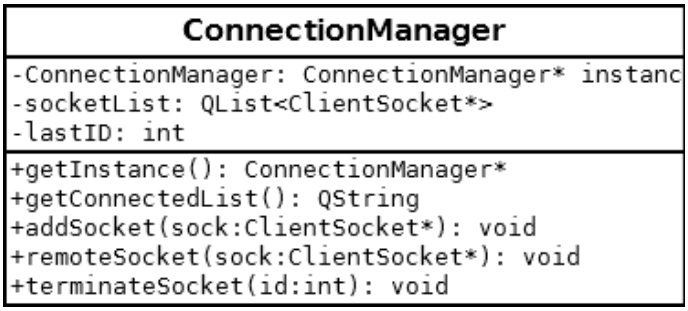
Obrázek 18: Třída Database

Poslední částí je seznam lokálních zařízení. Každé lokální zařízení je ukazatel na jednu instanci třídy LocalOutput, která bude dále popsána v kapitole 5.3.9. Samotný seznam je stejně jako u předchozího případu kolekce typu QList tentokrát zvaná localOutputList.

5.3.7 Třída Database

Jedná se o třídu, která uchovává všechny informace potřebné k tomu. Tyto informace jsou přihlašovací jméno a heslo k databázi, název databáze, typ databáze a IP adresa umístění databáze, stejně jako je zakresleno na obrázku 18. Dále jsou ve třídě tři metody, které všechny vracejí datový typ bool. V tom je obsažena informace, zda vše proběhlo v pořádku či nikoli. Problému mohou nastat buď v samotném otevření komunikačního kanálu (metoda open) nebo v samotném zpracování dotazu, například neexistující tabulky.

Jsou dále k dispozici dvě metody. První z nich, metoda getDatabaseHash, se dotáže databáze na hash hesla konkrétního uživatele. Samotný hash pak vrací jako ukazatel na argument metody. Druhou metodou je vypsání všech zařízení v konkrétní rezervaci. Opět podobně jako v předchozí metodě vrací odpověď jako ukazatel na řetězec argumentu metody.



Obrázek 19: Třída ConnectionManager

5.3.8 Třída ConnectionManager

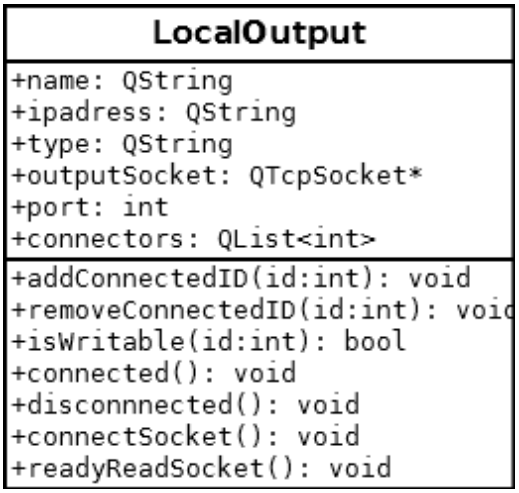
Jedná se o druhý sigleton použitý v aplikaci. Zde je použití sigletonu jediná možnost, jak mohou být sdíleny všechny spojení v celé aplikaci (příchozí spojení pro standardní uživatele a uživatele s právy administrátora je až příliš programově oddělena). Stejně jako v předchozím případě tu máme instanci i konstruktor v privátní části a metoda getInstance je ve veřejné.

Důležitý je seznam všech spojení, který je kolekcí všech instancí ve třídy ClientSocket, které se vytvářejí při žádosti o nové spojení. Do této kolekce se přidávají metodou addSocket, ve které se jí také přidělí identifikátor, který je odvozen od hodnoty lastID, tj. naposledy použitému identifikátoru. Pro odebrání socketu z kolekce slouží metoda removeSocket, která se vyvolá vždy, když instance třídy ClientSocket vyvolá signál exitSocket, která značí zrušení dané instance. Dále se porovná počet všech spojení a je-li rovno nule, tak se nastaví lastID také na nulu. To je k dosažení toho, aby časem tento identifikátor posledního spojení nepřetekl přes své maximum a nezpůsobil chybu v aplikaci. Je k tomu i připočten lidský faktor který předpokládá, že nastane doba, kdy k serveru nebudou připojeni žádní uživatelé (noční hodiny).

Nakonec jsou tu metody getConnectedList a terminateSocket, které obě slouží pro příkazy, ke kterým má přístup administrátor. První z nich vypíše všechna spojení tím, že z každé instance si vyžádá její popis přes metodu getDescription. Ty spojí dohromady a následně je pošle zpět jako návratovou hodnotu. Druhou metodou je terminateSocket. Ta otestuje ve identifikátory všech spojení a je-li shoda, pak u této instance zavolá metodu terminate. Třída je zakreslena na obrázku 19.

5.3.9 Třída LocalOutput

Třída reprezentující spojení k odchozímu zařízení či konzolovému serveru v jiné lokalitě. Vzhledem k tomu, že každé zařízení může obsluhovat více uživatelů (více spolupracovníků v jedné rezervaci), ale ke každému zařízení může vést pouze jedno spojení, je nutné aby komunikaci se zařízením obsahovala tato třída. Jak je ukázáno na obrázku 20 obsahuje objekt typu QTcpSocket, který je spojení se zařízením. Není-li žádný uživatel,



Obrázek 20: Třída LocalOutput

požadující spojení se zařízením, je hodnota ukazatele na tento objekt NULL. Objekt se instanciuje pouze v případě, že je požadováno spojení se zařízením.

Dále je potřeba instanci objektu nějak obsluhovat. K tomu slouží metody connected, readyReadSocket, disconnected a signál dataToRead. Tyto metody komunikují se všemi instancemi třídy ClientSocket, které si komunikaci propojily. Metoda connected sděluje, že odchozí spojení je připraveno ke komunikaci a je na řídících parametrech, aby buď začala posílat data od uživatele (připojení k lokálnímu zařízeni) nebo dále čekat (na odpověď od vzdáleného konzolového serveru, který musí sám navázat komunikaci se zařízením). Metoda disconnect pouze nastaví outputSocket na NULL a vyprázdní seznam identifikátorů všech spojení. Metoda readyReadSocket se vyvolá tehdy, když jsou na bufferu socketu připraveny data ke čtení. Ty se argumentem signálu dataToRead pošlou všem připojeným objektům a následně zpracují (odešlou příslušným uživatelům).

Díky omezení zasílání dat na zařízení jen jednomu uživateli, existuje ve třídě kolekce connectors, která obsahuje seznam identifikátorů všech připojených instancí třídy ClientSocket. K práci s touto kolekcí existují tři metody: addConnectedID, removeConnectedID a metoda isWritable. První jednoduše do kolekce přidá nový identifikátor, který přijde v argumentu metody. Druhá metoda pak tento identifikátor odebere, ale navíc pokud již je kolekce prázdná tak se provede odpojení spojení s druhou stranou. Poslední metodou je metoda isWritable, která jen identifikátor argumentu porovná s vrcholem kolekce (v tomto případě typu FIFO) a vrátí true pokud jsou shodné. Čili zapisovat na zařízení může jen první uživatel, který zažádal o připojení nebo v případě, že se již odpojil tak uživatel, který se připojil další v pořadí.

5.4 Implementace klientské aplikace

Klientská aplikace je druhá část programovací části této diplomové práce. Její existence by měla usnadnit přístup k zařízením a jejich následnou konfiguraci. Samotná aplikace je

napsána ve frameworku Qt, což umožňuje kompilaci pod všemi hlavními platformami. Samotná aplikace je rozdělena na tyto části:

- **Hlavní okno**
- **Dialogové okno pro nové spojení**
- **Dialogové okno pro výběr zařízení**
- **Konkrétní záložka komunikace se zařízením**

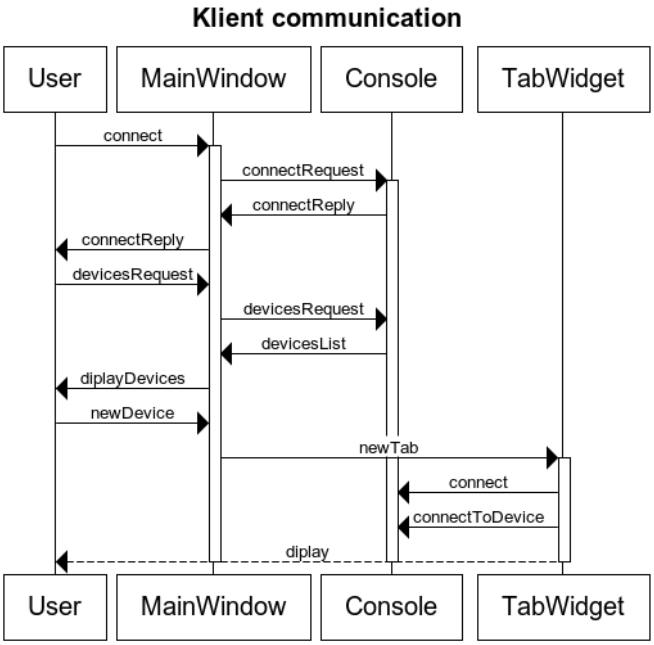
Hlavní dialogové okno je tvořeno třídou MainWindow. Jejím účelem v aplikaci je udržovat si všechny další grafické i konfigurační prvky. Obsahuje tři hlavní části: menu, nástrojovou lištu a centrální prvek. Menu je tvořeno odkazy na dialogové prvky aplikace stejně jako nástrojová lišta, která umožňuje snadnější přístup k těmto prvkům. Centrální prvek je tabularizované (tvořené záložkami), který je tvořen prvkem QTabWidget.

V dialogovém okně pro nové spojení se zadávají informace pro připojení ke konzolové aplikaci. Zadává se jen uživatelské jméno, heslo a adresa serveru. Samotné ověření správnosti probíhá ve třídě MainWindow, kdy se kontroluje jak vlastní možnost připojení k serveru (nesprávně zadaná adresa nebo nedostupnost konzolového serveru) tak i správnost zadaného hesla (kontrola podle hodnoty, kterou přijme z konzolového serveru). Všechny údaje se ukládají a načítají z/do souboru, takže všechny údaje je nutné zadat pouze jednou. Tím se zrychlí přístup k hlavnímu použití aplikace (komunikace se zařízeními).

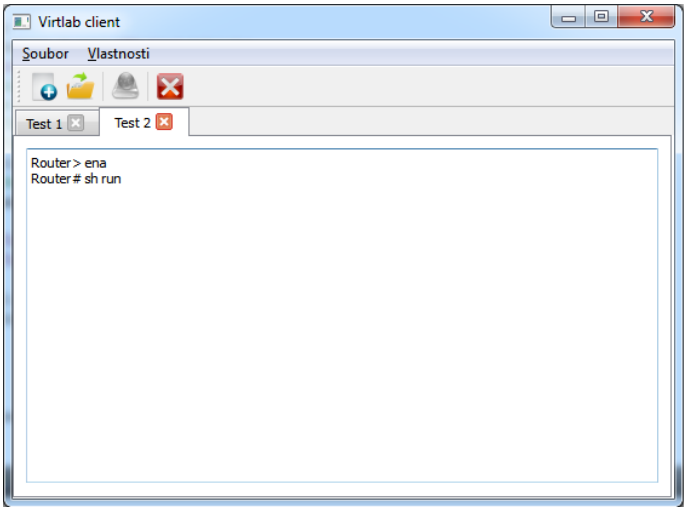
Při kliknutí na tlačítko Připojení k zařízení se nejprve pošle příkaz 'list' do již otevřeného a ověřeného spojení s konzolovým serverem. To vrátí seznam všech možných zařízení, ke kterému se může uživatel připojit. Tento seznam se dále pošle do dialogového okna, ve kterém se zpracuje a zobrazí se ve vícevýběrovém seznamu. Každý prvek se zobrazí pod svým názvem a identifikátorem (příklad R1 - r1@vsb-advanced.cz). Seznam je vícevýběrový což znamená, že je možno vybrat najednou více prvků a po kliknutí na tlačítko ok se všechny otevřou. Dále při dvojkliknutí na prvek se k již otevřeným záložkám přidá nová s vybraným prvkem. Záložky jsou pojmenované podle názvu prvku.

Poslední částí je záložka s konkrétním zařízení. Každá záložka je tvořena instancí třídy TabWidget, která v sobě obsahuje socket. Do konstuktoru je poslána instance nastavení, ze které se vyberou údaje, které byly použity při prvním spojení a socket s pomocí těchto údajů naváže komunikaci s konzolovým serverem. Dále po autentizaci pošle příkaz 'connect' s parametrem unikátního identifikátoru zařízení a čeká na odpověď. Přijde-li mu řetězec 'ok' pak povolí interakci s textovým editorem, který slouží jako vstupně/výstupní zařízení pro zadávání příkazů. Samotné spojení s klientem je neblokující, ale následné zadávání uživatelských dat je kvůli jednoduchosti zadáváno v blokujícím režimu, takže se může stát, že aplikace může nějakou chvíli čekat, než ji přijde odpověď.

Komunikace celé aplikace je zakreslena na sekvenčním diagramu na obrázku 21. Vlastní aplikace pod prostředím Microsoft Windows je na obrázku 22.



Obrázek 21: Komunikace klientské aplikace



Obrázek 22: Ukázka klientské aplikace v prostředí Windows

5.4.1 Chování textového editoru

Jelikož textový editor se chová odlišněji než konzole, bylo potřeba jeho chování upravit, aby ji co nejvíce připomínala. Odlišnosti a specifika konzole jsou následující:

- Šipky doleva a doprava - v konzoli se pohybují jen v oblasti jednoho řádku a to jen za promptem (výzvou). V textovém editoru umožňuje pohyb od začátku až do konce.
- Šipky nahoru a dolů - v konzoli slouží k zobrazení předchozích příkazů, v editoru k posunu kurzoru mezi řádky
- Ctrl - v konzoli slouží jako kontrolní tlačítko pro zvláštní příkazy. V editoru nemá zvláštní význam
- Tab - v konzoli složí jako našeptávač pro právě psané příkazy. V editoru jen jako odsazení textu.
- Enter - v konzoli funguje jako signál pro zpracování příkazu. V editoru jako odřádkování.

Všechny tyto odlišnosti musely být brány v potaz při zpracování aplikace. Nejjednodušším způsobem, jak je zpracovat bylo vytvoření nové třídy MyTextEdit, která dědila ze třídy QTextEdit, ale redefinovala dvě metody: keyPressEvent a keyReleaseEvent. Jak název napovídá, jsou to metody zodpovědné za zpracování stisknutých znaků na klávesnici.

6 Testování

Vzhledem k tomu, že aplikace nesmí obsahovat žádné chyby, bylo potřeba ji řádně otestovat. Řádné otestování bylo i jedním z bodů zadání diplomové práce. Celé testování se dá rozdělit do několika fází. Jsou to:

- Fáze 1 - testování samotné komunikace, fungování metod frameworku, funkce reakcí na změny v socketu apod.
- Fáze 2 - testování správného chodu jednotlivých funkcí systému (ověření identity, reakce na příkazy)
- Fáze 3 - testování komunikace multiplatformního klienta s konzolovou aplikací
- Fáze 4 - testování odezvy, stability aplikace konzolového serveru a obecné ujištění o správnosti funkce.

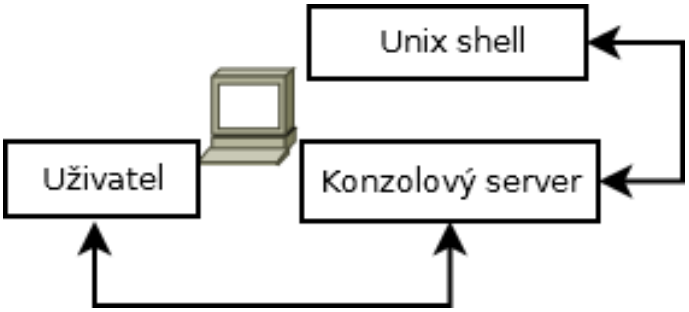
Tyto fáze přicházely postupně se samotným vývojem celé aplikace. Dalo by se říci, že pro vývoj byl použit alespoň částečně vodopádový model vývoje, jelikož před každou implementací nové funkcionality probíhala analýza a po implementaci probíhalo následné testování.

V první fázi se řešila samostatná komunikace s konzolovým serverem. V základu byla postavena jednoduchá aplikace, která pouze naslouchala a vypisovala statický text po přijetí jakýchkoliv dat. Vývoj probíhal na operačním systému Kubuntu 12.10. Pro samotnou komunikaci s konzolovou aplikací bylo potřeba jednoduchého klienta, postačující byl jakýkoliv klient pro síťovou komunikaci s textovým výstupem. Pro potřeby byl tedy vybrán počítačový program s názvem netcat, který již byl v operačním systému Kubuntu obsažen a bylo jednoduché ho použít. Samotné navázání komunikace s konzolovým příkazem pak probíhalo následujícím příkazem:

nc localhost [číslo portu]

V původním plánu bylo předpokládáno s použitím vláken, se kterými byla i tato jednoduchá aplikace (konzolový server ve fázi 1) napsána. Bohužel, se při použití vláken vyskytly chyby, které nebylo možno odstranit. Jednalo se o podivné reakce na příchozí zprávy, kdy při zaslání dat aplikace nereagovala tradičním vyvoláním signálu readyRead, nebo se při zasílání dat data poslala jednou, dvakrát nebo dokonce vůbec. Celá tato komunikace byla pro svoji zvláštnost neustále zachycována v aplikaci Wireshark (populární protokolový analyzér a paketový sniffer), aby se otestovalo, zda se data vůbec zasílají (a tím vyloučit chybu programu Netcat). Bohužel se chyba aplikace Netcat vyloučila a bylo rozhodnuto upustit od přístupu s vlákny, což ve výsledku znamenalo zlehčení práce při programování dalších částí kódu (potenciální nutnosti zamykání apod.). Pro další implementaci bylo tedy vybrána obyčejná asynchronní komunikace se sokety.

Ve druhé fázi se testovala prvotní komunikace s aplikací v rámci její funkce. Ze všeho nejdřív pak autentizace uživatele a jeho reakce na příkazy. Hned v autentizaci, kdy se přistupovalo k databázi virtlabu byla nalezena nemožnost připojení se k běžící instanci. Po



Obrázek 23: Komunikace s unixovým shellem jako zařízení

testování bylo zjištěno, že aktuální nastavení databáze neumožňovala vzdálený přístup, a tak pro potřeby testování bylo toto upraveno v konfiguračním souboru databáze mysql a po restartu databáze již komunikace probíhala. Dále v této fázi probíhalo testování jednotlivých příkazů a tedy již prakticky celá základní funkčnost aplikace. Pro potřeby testování připojení se k zařízení bylo použito testování na lokálním zařízení bez použití fyzických prvků. K tomu bylo využito přesměrování lokálního unixového shellu na naslouchání vybraného portu. Prakticky se to docílilo následujícími příkazy:

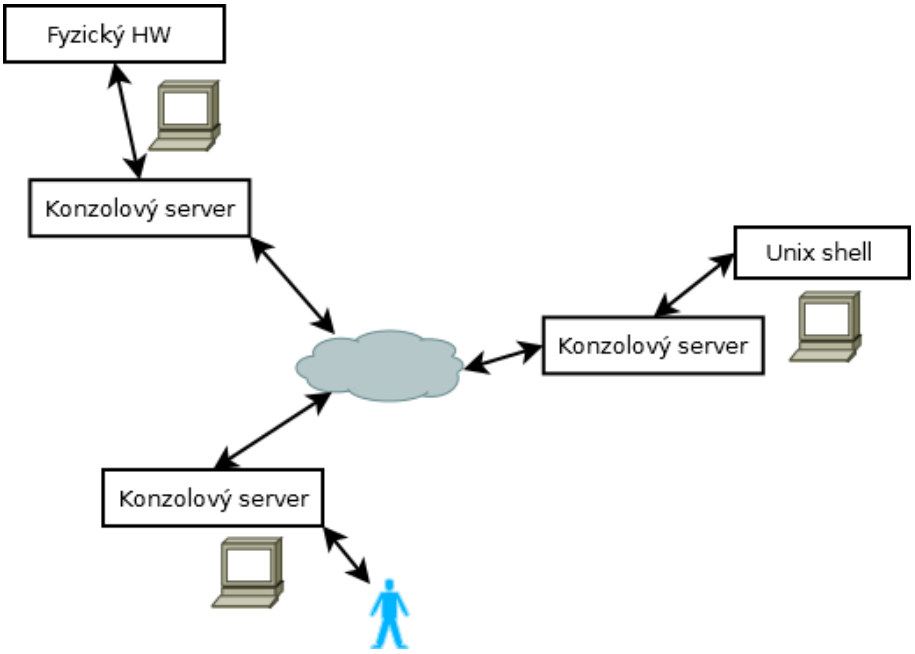
```
mkfifo /tmp/mypipe
```

```
cat /tmp/mypipe|/bin/bash 2>&1|nc -l 6000 >/tmp/mypipe
```

Díky těmto příkazům se unixová konzola zobrazí všem uživatelům, kteří se připojí na port 6000 (naslouchající port se nastaví za argumentem -l). Dále bylo vytvořeno nové zařízení v konfiguračním souboru se zařízeními, byl mu dán zvolený port a jako IP adresa mu byla dána adresa localhostu (127.0.0.1). Komunikace pak probíhala, jak je zakresleno na obrázku 23. V této fázi se objevilo nejvíce chyb a nejvíce chyb bylo ošetřeno.

Ve třetí fázi byla hotová multiplatformní aplikace a bylo potřeba otestovat její komunikaci s konzolovou částí. Hlavně pak bylo potřeba vyřešit problém víceuživatelského přístupu k jednomu zařízení, což bylo potřeba vyřešeno sdílenou komunikací mezi vlastním zařízením a konzolovým serverem. Samotná multiplatformní aplikace musela komunikovat s konzolovou aplikací nejen pro napojení k zařízením, ale i k získání informací o dostupných zařízeních a možnost autorizovat uživatele, který aplikaci používá. Při samotné komunikaci nebyl větší problém, pouze se muselo použít čekání na příchozí data, před odesíláním dat, které jsme chtěli na server zaslat. Avšak došlo ke stavu, kdy ukončení aplikace způsobilo pád samotné konzolové strany, což rozhodně nebyl stav, kterého by se mělo dosáhnout. Proto bylo vše pečlivě odladěno a chyba ošetřena.

V poslední fázi se dá říci, že celá aplikace je hotova. Je pouze potřeba ošetřit krizové stavy a otestovat její celkovou stabilitu. Pro tuto fázi bylo provedeno zapojení, které je zakreslené na obrázku 24. Ještě před samotným testováním bylo doprogramováno do aplikace uvolňování paměti, ať nedojde k zahlcení fyzických prostředků (v předchozích fázích toto nebylo potřeba, aplikace nikdy neběžela dost dlouho na to, aby zabrala v paměti příliš místa). Byla také relativně testována doba odezvy jednotlivých prvků, aby



Obrázek 24: Testování ve fázi 4

se vyloučilo příliš velké zpoždění od zaslání příkazu po dobu přijetí odpovědi, jejího následného zpracování a vykreslení na grafický či textový výstup. Naštěstí aplikace pracovala svižně (v relativním pojetí). Dále se testovaly reakce aplikace v případech ztráty spojení z důvodů nefunkčnosti fyzického spojení. Bylo také dodáno zpracování příkazů, které vyžadují některé prvky pro jejich jednodušší ovládání.

V každé fázi byly nalezeny chyby či výjimky, které bylo nutné ošetřit či zpracovat. V některých případech došlo i k přepsání velké části zdrojového kódu tak, aby aplikace byla co nejvíce stabilní a použitelná v reálném provozu. Nevylučuji však možnost výskytu chyby při podmínkách, které se nedají snadno otestovat (extrémní zátěž serveru, zatížení linky, velký jitter u spojení apod.).

7 Připravení aplikací k použití

Spuštění aplikace jako takové je pro finální nasazení nedostatečné. Aby bylo možno hotovou aplikaci dát k dispozici jsou potřeba dva kroky:

1. Zkompilování aplikace v release módu, kdy se do kódu dostanou jisté optimalizace a naopak se odstraní části, které jsou nutné k debuggování.

2. Dodání externích knihoven

Vzhledem k tomu, že Qt není systémová knihovna, je potřeba aby byla redistribuována společně s aplikací. Pro distribuci je postačující přibalit knihovny, které aplikace potřebuje do adresáře, kde se spustitelný soubor nachází (na platformě Microsoft Windows). Tyto knihovny jsou specifikovány uvnitř projektového souboru a pro klientskou aplikaci jsou potřeba následující knihovny:

- **QtGui**
- **QtCore**
- **QtNetwork**

Vzhledem k tomu, že konzolová aplikace má běžet na operačním systému Debian, nebo systému z něho vycházejícího, je možno využít jejího balíčkového systému (tj. filosofie, že každý program má svůj balíček, který se dá představit jako archív). Tedy je možné dodat do systému balíček obsahující tyto knihovny do systému dříve, než se spustí samotná instance konzolového serveru. To se dá provést například tímto příkazem:

apt-get install libqt4

Nebo je možno vytvořit samostatný balíček, který veškeré knihovny bude obsahovat a který je při spuštění automaticky nahraje do systému.

Tento systém, kdy se knihovny načítá dynamicky, poskytuje výhodu například snížení nároků na paměť, šetření prostoru, jednodušší aktualizace (s novou verzí knihoven se nemusí překompilovat vlastní aplikace). Proto bylo přistoupeno k tomuto systému vydání programové části hotové diplomové práce.

8 Závěr

Cílem práce bylo vytvořit novou formu konzolového serveru tak, aby se stávajícím řešením bylo v základní funkčnosti stejné, tj. aby dokázalo bezproblémově spojovat uživatele se zařízení, a navíc, aby obsahovalo jakousi formu zjednodušení přístupu na toto nové řešení. Dále vzhledem k použitému objektovému přístupu při psaní kódu samotné aplikace byl dbán důraz na to, aby mohla být aplikace snadno rozšířena, pokud budě někdy v budoucnu požadavek na novou funkcionalitu či vylepšení nového řešení.

Dále práce poskytla základní náhled do distribuovaných a nedistribuovaných verzí virtuální laboratoře, která sloužila studentům k plnění studijních projektů v rámci výuky nejen v rámci Cisco vzdělávacích programů, ale i k výuce počítačových sítí na Vysoké škole báňské a i na jiných univerzitách, které se na projektu podílely. Byly stručně popsány jednotlivé prvky distribuované verze a dále jako její součást i konzolový server, který samotný byl cílem práce.

Větší část práce byla věnována rozboru síťové komunikace, vzhledem k tomu, že se jedná o něco, na co je v praktické (programovací) části práce dáván velký důraz a je pro pochopení samotného programu nezbytné mít znalosti i z této oblasti. Byl popsán základní kámen komunikace v počítačových sítích, lehce naznačen vrstvový model a největší důraz této sekce byl dán na navazování spojení a s ní spojené další věci, které se dále použily do vlastní práce. Nakonec byl rozepsáno o použití těchto způsobů síťové komunikace na softwarové úrovni ve formě programovacího API zvané sockety.

Jak bylo specifikováno i v zadání práce, byla potřeba podrobná analýza dané problematiky. To znamená, že bylo nutné se seznámit nejen se stávajícím řešením komponenty konzolového serveru, ale i dalších komponent distribuované virtuální laboratoře. Část této sekce byla věnována databázi, která je postavena na variantě MySQL. Byl proveden rozbor jednotlivých tabulek, které se v samotné aplikace dají použít nebo bylo předpokládáno jejich použití. Analýza pak navíc obsahuje ještě popisy komunikace mezi uživatelem a samotnou aplikací, kdy bylo potřeba vybrat všechny užitečné funkcionality pro uživatele a navrhnout systém autentizace tak, aby byl co nejméně rušivý pro externí použití (tj. skrze externího klienta). Nakonec se tato sekce analýzy věnovala přístupu k zařízením a vybrání nejoptimálnější varianty přístupu více uživatelů, kdy byl vybrán přístup, který omezuje všechny až na jednoho uživatele jen režimem čtení zaslaných dat. Vybraný uživatel má pak jako jediný právo zápisu.

Nejdůležitější částí byla samotná implementace. Byly popsány jednotlivé možnosti při výběru prostředí, ve kterém bude psán kód a zdůvodnění výběru frameworku Qt. Dále byla celkem podrobně popsána implementace samotné konzolové části s pečlivě vypsanými vztahy, mezi jednotlivými částmi. Byla také naznačena implementace klientské části, která byla dalším bodem zadání diplomové práce. Jedním z požadavků byla její multiplatformnost, čehož bylo nakonec docíleno vytvořením nativních aplikací pro platformy Microsoft Windows, Linux a Mac OS X.

Nakonec neméně důležitou částí a posledním bodem zadání bylo otestování aplikace. Bylo popsáno kladení důrazu na neustále testování funkčnosti, výsledné testování stability a rozdělení do několika testovacích fází. Byly poskytnuty schémata, na kterých se

testovalo. Navíc bylo popsáno i samotné vydání aplikace, co vše musí aplikace obsahovat, aby byla na samotných platformách spuštěna.

Výsledné řešení bylo pečlivě otestováno a ošetřeno na případné chyby, které se v průběhu vývoje mnohokrát objevili. Není však vyloučena existence chyby, protože dokonalé testování není možné a další vyloučení chyb je nutné zjisti v ostrém provozu.

Matěj Děcký

9 Reference

[1] Beck, Michael *Linux kernel internals 2nd,* USA: Addison-Wesley, 1998.

[2] Benvenuti, Christian *Understanding Linux network internals*, USA: O Reilly Media, 2006.

[3] Smith, Roderick W. *Advanced Linux networking*, USA: Addison-Wesley, 2002.

[4] Vetle, Toby J., Vetle, Anthony *Síťové technologie Cisco*, Computer Press, 2003.

[5] DIGIA. *Qt* [online]. 2013 [cit. 2013-04-25]. Dostupné z: http://qt.digia.com/

[6] Projekt VirtlabWiki. *Virtuální laboratoř počítačových sítí* [online]. 2012 [cit. 2012-08-16]. Dostupné z: http://www.cs.vsb.cz/vl-wiki/index.php/Virtuální_laboratoř_počítačových_sítí